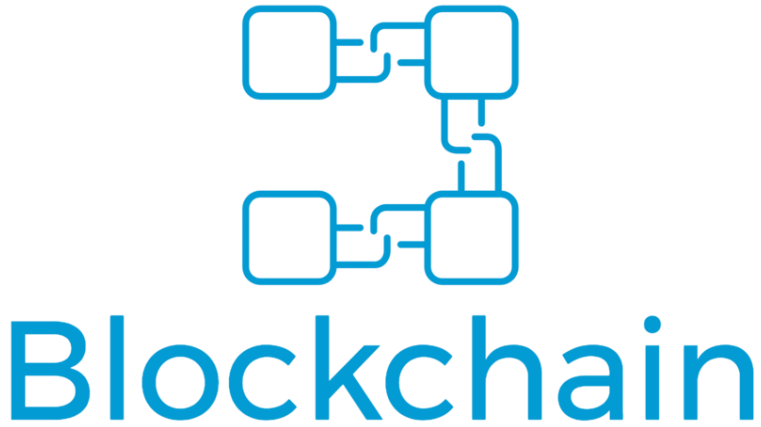
Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

La blockchain pour réaliser du recommandé

Rapport

Projet de semestre 5

Filière Télécommunication – Internet et communications



Maxime Genilloud - Damien Morattel

T3F

Responsable externe : Jonathan Gilgen – Softcom

Responsables internes : Houda Chabbi
Patrick Gaillet

Version 1.0 du 31.01.2019

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Table des matières

1 Introduction 4

2 Analyse 5

2.1 Blockchain 5
2.1.1 Critères de sélection 6

2.2 Tableau comparatif 8

2.3 dApps 9

2.4 Smart Contracts 10

2.5 Hyperledger 12

2.6 Hyperledger Composer 13

2.7 Solutions 14
2.7.1 Utilisation d'une blockchain publique ou privée 14
2.7.2 Multicomptes sur la blockchain 14

2.8 Notre solution 15

3 Spécifications 16

3.1 Fonctionnement global 16
3.1.1 Limites de notre Proof-of-Concept 17
3.1.2 Etats de l'envoi 17

3.2 Use Case 18

3.3 Diagramme des composants 18

3.4 Blockchain 19
3.4.1 Données stockées 19
3.4.2 Smart contracts 19

3.5 Back-End 20

3.6 Front-End 20

4 Réalisation 21

4.1 Infrastructure Blockchain 21

4.2 Hyperledger Composer 21
4.2.1 Modélisation 22
4.2.2 Transactions 23
4.2.3 Requêtes 24
4.2.4 Évènements 25
4.2.5 API REST 25
a) Appels 26
b) Retours 27

4.3 Back-End 28
4.3.1 Authentification des utilisateurs 29
4.3.2 Interactions avec la blockchain 31
4.3.3 Gestions des erreurs 32
4.3.4 Diagramme de séquence 33

4.4 Front-End ... 34
4.4.1 Interface utilisateur ... 34
4.4.2 Composants ... 38
4.4.3 Modèles ... 40
4.4.4 Services ... 41
4.5 Problèmes rencontrés ... 44
4.5.1 CORS ... 44
4.5.2 MVCC_READ_CONFLICT ... 44
4.6 Futur et améliorations ... 45
4.6.1 Contrôles niveau Blockchain ... 45
4.6.2 Front-end mis à jour en temps réel ... 45
4.6.3 Authentification des utilisateurs ... 45
4.6.4 Trier les documents par date d'envoi ... 45
4.6.5 Traitement des erreurs ... 46
4.6.6 Développement natif sur Hyperledger Fabric ... 46
5 Rendu ... 47
5.1 Structure des dossiers ... 47
6 Conclusion ... 48
6.1 Atteinte des objectifs ... 48
6.1.1 Rapport d'analyse ... 48
6.1.2 Use case ... 48
6.1.3 Prototype ... 49
6.2 Conclusions personnelles ... 49
6.2.1 Maxime Genilloud ... 49
6.2.2 Damien Morattel ... 49
7 Déclaration d'honneur ... 50
8 Glossaire ... 51
9 Références ... 52
9.1 Figures ... 52
9.2 Tableaux ... 53
9.3 Bibliographie ... 54
10 Annexes ... 56

1 Introduction

Dans la société actuelle, il arrive régulièrement que des documents importants doivent être transmis d'une entité A à une entité B. Cela peut être par exemple un contrat, des résultats d'examens ou encore une assignation à comparaître.

Pour l'entité A, il est primordial de pouvoir certifier la réception des dits documents par l'entité B. Le courrier recommandé répond à cette problématique. En effet, un courrier recommandé ne peut être délivré qu'à son destinataire contre vérification de son identité et sa signature. Un justificatif de distribution est alors transmis à l'entité A. Ce papier a une valeur légale. La Poste agit comme tiers de confiance et se charge de délivrer le courrier et d'attester de sa bonne réception. Pour cela, l'entreprise est rémunérée (entre 5,30 CHF et 6,30 CHF).

Dans certains cas, un envoi postal n'est pas forcément adéquat. En effet, celui-ci prend au moins un jour ouvrable pour être acheminé (dans le cas d'un recommandé). De plus, certaines données sont plus pratiques à traiter numériquement. Même si une clé USB peut être transmise par courrier, ce n'est pas le moyen de transport idéal (accident).

Notre projet cherche donc à adapter le courrier recommandé à l'ère du numérique. L'idée est de permettre aux deux parties de se transmettre des documents numériques via une plateforme dédiée. Afin d'envoyer les documents et d'attester de leur bonne réception, la blockchain sera utilisée comme tiers de confiance.

Cela permettrait ainsi de réduire grandement le temps nécessaire à l'envoi d'un recommandé. De plus, l'envoi serait possible à tout moment. Finalement, les coûts seraient fortement réduits.

2 Analyse

Le chapitre suivant aborde les différentes recherches que nous avons effectuées afin de trouver une solution optimale pour notre projet.

2.1 Blockchain

Une blockchain est un livre de comptes décentralisé, composé de blocs de données chainées. Le tout repose sur différents algorithmes cryptographiques permettant une sécurisation du réseau.

Les utilisateurs d'une blockchain interagissent avec cette dernière en générant un compte lié à une clé privée leur permettant d'obtenir une balance d'avoir. Ce compte leur permet d'envoyer et recevoir des transactions à d'autres comptes. Chaque compte est représenté par une chaine de caractères unique, servant d'adresse de réception pour le compte. Les transactions enregistrées sur la blockchain peuvent également contenir des données.

Une blockchain est composée des éléments suivants :

Block	Un block est un ensemble de transactions. Les blocks sont limités à une certaine taille. Une fois remplis, ils sont chainés et font donc partie de la blockchain.
Noeud	Ordinateur/serveur faisant partie du réseau blockchain. Il utilise un programme relayant les transactions. Les nœuds possèdent tous une copie complète du registre de la blockchain.
Transactions	Une transaction est créée lors de l'appel d'un smart contract. Elle contient les données ajoutées ou modifiées de la blockchain. Les transactions sont validées par les mineurs du réseau avant d'être insérées dans le dernier block.
Mineur	Un mineur est un utilisateur qui apporte la puissance de calcul au réseau blockchain. Il s'occupe de vérifier si les transactions sont correctes pour les valider et les ajouter au dernier block. Ce mécanisme fonctionne sur la base de calculs cryptographique. Les mineurs sont généralement rémunérés pour leur travail grâce aux frais de transactions.

Tableau 1 : Eléments constituants une blockchain

2.1.1 Critères de sélection

Type de blockchain
Il existe trois types de Blockchain :

- **Publique**
  Une Blockchain publique peut être consultée et écrite par n'importe qui dans le monde. Elle est complétement décentralisée et tout le monde peut mettre en place un nœud. C'est le premier type de blockchain qui est apparue. L'avantage de ce type est la complète décentralisation et protection contre les attaques. Mais le désavantage est la vitesse et l'évolutivité du réseaux

- **Privée**
  Ce type de blockchain est accessible uniquement par les personnes autorisées au préalable par le créateur du réseau. Ce type de blockchain est adapté à une utilisation interne à une organisation. L'avantage est la rapidité, par contre le réseau n'est plus vraiment décentralisé et donc plus vulnérable aux attaques.

- **Consortium**
  Sorte d'hybride entre publique et privée, ce type de blockchain, aussi appelée permissionnée ou fédérée, est contrôlé par un groupe présélectionné de nœuds. Ce type de réseaux est adapté à un groupe de sociétés travaillants ensemble et échangeant des données.

Consensus
Le consensus est la procédure grâce à laquelle un réseau de nœuds va pouvoir garantir les transactions. Il existe actuellement plusieurs implémentations permettant la vérification de validité des transactions générées par les utilisateurs avant d'intégrer ces dernières à la blockchain.

La plus utilisée et connue est la preuve de travail (proof of work). Cette méthode repose sur le travail des mineurs. Les mineurs utilisent un logiciel propre à la blockchain qui s'occupe de vérifier l'exactitude des transactions. Afin qu'une transaction soit intégrée à la blockchain, il faut que plus de 50% des mineurs soient d'accord sur la validité de cette dernière. Les mineurs sont rémunérés en avoir sur la blockchain. Cet avoir est issu des frais liés à chaque transaction. Le gros désavantage de cette méthode de vérification est la consommation électrique générée par les mineurs et donc son coût.

Une autre méthode de vérification a fait son apparition. Il s'agit de la preuve d'enjeu (proof of stake). Cette méthode permet à l'utilisateur de bloquer sa balance sur la blockchain (mise en jeu) afin d'obtenir un nombre de vote proportionnel au montant qu'il a bloqué. Ces votes sont ensuite utilisés pour valider les transactions. Cette méthode part du principe que les personnes ayant beaucoup d'avoir sur la blockchain ont plutôt intérêt à supporter cette dernière en votant pour des transactions correctes, sans quoi ils risqueraient de perdre leurs avoir. Un intérêt est perçu sur l'avoir mis en jeu grâce, encore une fois, aux frais de transactions des utilisateurs.

Il existe d'autres consensus plus marginaux. On peut par exemple citer DPoW (Delegated Proof of Work) et DPoS (Delegated Proof of Stake), qui fonctionne sur la même base que leurs parents, à l'exception près qu'un groupe d'utilisateurs est désigné par la communauté comme représentants (super representative) et ce sont eux qui vont se charger de mettre en œuvre l'algorithme de cryptage.

Temps d'un bloc

Le temps de cryptage d'un bloc correspond au temps maximal théorique nécessaire pour qu'une transaction soit validée sur la blockchain. Si la transaction est réalisée en début de bloc, son temps de validation va être plus long que si celle-ci intervient vers la fin d'un bloc.

Frais de transaction

Suivant la blockchain, des frais de de transaction plus ou moins élevés peuvent être appliqués. En général, ces frais sont calculés selon la charge du réseau (rapport entre le nombre de transactions à traiter et le nombre de nœuds sur le réseau). Ils peuvent être très variables et sont adaptés après un certain temps, suivant la blockchain.

De plus, certaines blockchains appliquent des frais différents pour les appels aux fonctions des smart contracts.

Frais de déploiement d'un smart contract

Bien souvent, des frais sont perçus pour le déploiement d'un smart contract sur la blockchain. D'après la technologie, ces frais peuvent êtes perçus selon l'espace nécessaire pour stocker le code sur la blockchain, de manière fixe (frais fixe) ou autre.

Ces frais sont facturés pour chaque déploiement du smart contract. Ainsi, si une modification dans le code doit être opérée, des frais seront à nouveau perçus.

Année de création

L'année de création de la blockchain peut être un indice de fiabilité de celle-ci. En effet, la technologie blockchain étant actuellement encore très récente, son ancienneté est plus ou moins synonyme de son avancement. Le développement de la technologie est continu et appuyé par sa communauté.

Taille de la communauté

Il est intéressant de prendre compte de la taille de la communauté qui soutient une blockchain. En effet, cela peut donner une indication quant à sa popularité, la confiance accordée en une blockchain, ainsi que sur les possibilités de support de la part de la communauté (forums, tutoriels, etc.).

2.2 Tableau comparatif

Afin d'avoir une vision globale rapide des différentes blockchains auxquelles nous nous sommes intéressés, nous avons réalisé un tableau comparatif. Celui-ci est basé sur les différents critères énoncés précédemment.

Blockchain	Ethereum	EOS	NEO	Stellar	Komodo	Hyperledger (Fabric)	Hyperledger (Sawtooth)
Type de blockchain	Publique	Publique	Publique	Publique	Publique	Consortium / Privée	Consortium / Publique
Consensus	PoW PoS mi-2019	DPoS	PoS	SCP (Byzantine)	DPoW	Kafka (plugins)	PoET (plugins)
Temps d'un bloc	15 sec	0,5 sec	15 sec	5 sec	60 sec	0-5 sec	0-5 sec
Frais de transactions	Très variables *0,043 $	Aucun limite de transactions	Aucun actuellement 0,1 - 1 GAS/system call	0,001 XLM *0,00026 $	0.0001 KMD *0.000124 $	Aucun	Aucun
Frais deploiement smart contract	32000 gas + 200 gas/byte **55,25 $	~ 120 EOS/MB RAM **137,05 $	100-1000 GAS *2605 $	Minimum 1 XML difficile à estimer	330 KMD/année *405,9 $/année	Aucun	Aucun
Langages smart contracts	Solidity	C, C++	Javascript, C#, Python, Java, Go	Javascript, Java, Go	C, C++	Java, Go, Node.js	Java, Go, Node.js
Année de création	2015	2018	2014	2014	2016	2017	2017
Taille de la communauté	++++	+++	+++	+++	+	+++	++
Évolutivité	+	++	+	++	+	+++	++++
Basée sur une monnaie	OUI (ETH)	OUI (EOS)	OUI (NEO)	OUI (XLM)	OUI (KMD)	NON	NON

Tableau 2 : Comparatifs des différentes blockchains

*calculé selon le cours du 07.11.2018
**pour un smart contract de 0,2 MB

2.3 dApps

Avant tout, il est difficile de donner une définition arrêtée de ce qu'est une dApp (Decentralized Application). En effet, il y a plusieurs visions qui gravitent autour de ce terme et tout le monde n'est pas d'accord sur le sujet.

Comme son nom l'indique, une dApp est une application décentralisée. Cela signifie que les données avec lesquelles l'application va interagir ne sont pas centralisées. Pour vulgariser, toute application basée sur une système pair-à-pair (P2P) peut être considérée comme une dApp.

Cependant, dans le cadre de la Blockchain, la dApp a une définition quelque peu différente. Elle prend le plus souvent la forme d'une application Web, remplaçant la base de données par la Blockchain et interagissant avec elle à l'aide de Smart Contract. Contrairement à une application Web standard où le code du Back End est centralisé sur des serveurs, le Back End d'une dApp est hébergé sur la Blockchain.

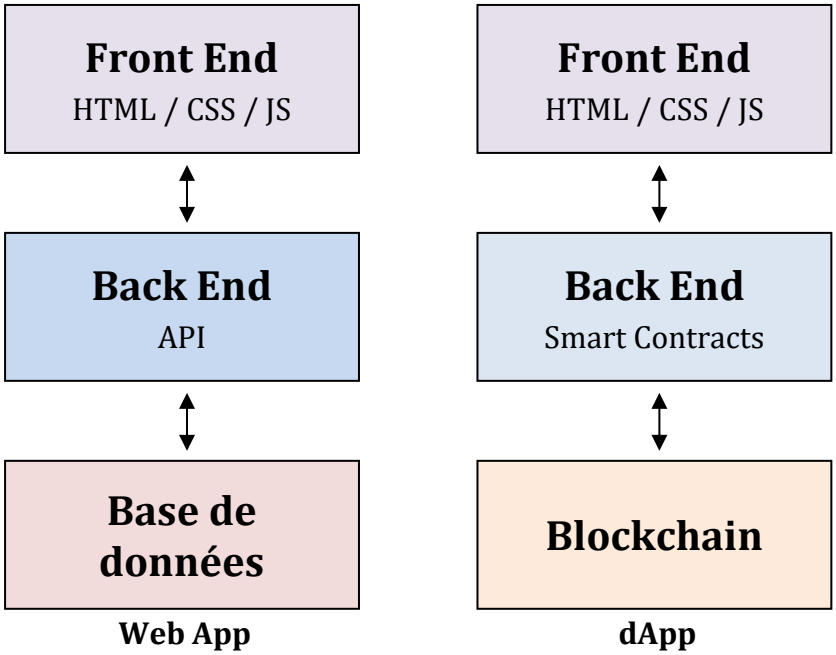


Figure 1 : Différence entre une application Web et une dApp

Il existe quatre critères qui sont généralement repris pour considérer une application comme étant une dApp :

- Le code source de l'application doit être **open-source** et géré de manière autonome (les modifications sont votées par consensus entre les utilisateurs).
- Les données de l'application doivent être enregistrées dans une **Blockchain**.
- Les mineurs doivent être récompensés à l'aide d'un **token**.
- Les tokens doivent être générés grâce à un **algorithme cryptographique**.

2.4 Smart Contracts

Un « smart contract », littéralement traduit par contrat intelligent, est un programme, ou plus simplement du code, stocké et exécuté sur une blockchain, de manière décentralisée.

On pense souvent que ce terme est apparu très récemment avec l'arrivée des blockchains comme Ethereum. Cependant, le concept de smart contract existe depuis 1995, année durant laquelle Nick Szabo, cryptographe et informaticien, écrit un article sur le sujet. Il faudra néanmoins attendre l'arrivée du Bitcoin en 2009 pour avoir une première implémentation du concept.

Le terme « contrat » est utilisé pour décrire ces applications car elles permettent d'automatiser des actions en fonction de règles (code informatique) pouvant représenter un accord entre plusieurs parties. En effet, on parle d'automatisation notamment car ces programmes permettent d'éliminer le tiers de confiance, qui dans le cadre d'un contrat physique, vérifie que les conditions du contrat sont remplies pour exécuter ce dernier.

Le second terme « smart » n'est pas bien choisi pour décrire les capacités de ces contrat digitaux. Effectivement, les smart contracts sont simplement du code informatique qui va prendre des décisions en fonction de différentes variables d'entrée et d'informations définies par le programmeur lors de la création du contrat.

La meilleure analogie pour représenter un smart contract est celle du distributeur de boissons. Un utilisateur insère de l'argent dans le distributeur puis tape un numéro pour faire sa sélection. La machine s'occupe automatiquement de vérifier que le montant introduit est suffisant et que la boisson est en stock avant de la délivrer. De la même façon, le smart contract vérifie que ses conditions soient validées avant d'exécuter des actions de manière automatique.

L'exemple du crowdfunding

Prenons l'exemple d'une plateforme de crowdfunding telle que Kickstarter. Cette entreprise permet le financement participatif de divers projets. Une équipe y propose un projet en indiquant comme objectif la somme d'argent qu'il souhaite réunir. Kickstarter permet à tout un chacun de verser de l'argent pour soutenir le projet. Si l'objectif est atteint dans un certain délai, l'équipe va recevoir cette somme. Dans le cas contraire, l'argent est restitué aux investisseurs. Dans ce cas, Kickstarter sert de tiers de confiance faisant le lien entre les supporteurs et la team du projet. Les désavantages de ce système sont la confiance en la tierce partie et les frais prélevés par cette dernière.

Un smart contract permet de créer un système de crowdfunding similaire de manière complètement décentralisée, qui délivre ou retourne les fonds automatiquement en fonction d'un code informatique. De cette manière, le tiers de confiance et les frais associés à ce dernier sont éliminés.

Fonctionnement sur la blockchain

Chaque blockchain a ses particularités et ses fonctionnalités. Certaines sont plus propices à gérer des smart contracts. Dans l'explication qui suit, nous allons nous baser sur la blockchain Ethereum qui a été conçue spécialement pour permettre la mise en place de smart contracts.

Nous savons qu'une blockchain est une sorte de base de données de transactions décentralisée et immuable. Les utilisateurs de la blockchain se créent des comptes sur cette dernière,

leur permettant d'envoyer et de recevoir des transactions contenant des données et/ou des avoirs digitaux grâce à une clé privée.

Dans ce contexte, les smart contracts peuvent être vus comme un diffèrent type de compte géré par un programme informatique et ne possédant pas de clé privée. Cela signifie qu'une fois le contrat déployé sur la blockchain, ce dernier est totalement autonome et personne ne peut en prendre le contrôle ni effectuer des transactions à la place du programme. La nature immuable de la blockchain empêche également toute modification du contrat. Ainsi, si une erreur s'y est glissée, il faudra déployer un nouveau contrat. Le smart contract, tout comme un compte standard, peut avoir une balance et effectuer des transactions.

Pour exécuter un smart contract stocké sur la blockchain, un utilisateur ou un autre contrat envoie une transaction à destination de l'adresse dudit smart contract en spécifiant quelle méthode de ce dernier il souhaite appeler, ainsi que d'éventuels paramètres. L'appel de cette fonction exécute les différentes instructions préalablement définies par le développeur. L'exécution est effectuée par les mineurs. Ces derniers comparent leurs résultats et s'ils correspondent, les changements sont propagés sur la blockchain.

L'entité ayant fait appel au smart contract doit payer des frais de transaction variants en fonction du nombre d'instructions et de leur complexité.

Limites
Tout ne peut pas être réalisé à l'aide de smart contracts. Certain cas d'utilisation sont plus faciles à implémenter que d'autres. Ces contrats intelligents ont leurs avantages, tels que la décentralisation, l'immuabilité et l'automatisme. Le problème actuel de ces contrats est le fait qu'il est impossible d'obtenir des informations du monde physique. En effet, un smart contract ne peut pas faire de requêtes hors de la blockchain. Le seul moyen de recevoir des informations externes est qu'une autre entité appelle le contrat en lui passant des informations en paramètre. Comme le contrat exécute des actions en fonction de ces informations, il faut avoir une confiance absolue en cette entité externe. Ceci compromet tout l'aspect décentralisé et "trustless" des smart contracts. Dans l'univers blockchain, une telle entité fournissant des informations sur le monde réel s'appelle un « oracle ».

Avantages et inconvénients
Avantages

- Immuabilité : Impossible de modifier un smart contract déployé.
- Décentralisation : Pas de tiers de confiance nécessaire.

Inconvénients

- Sécurisation : Il faut faire très attention à cause de l'immuabilité du code.
- Difficile d'interagir hors de la blockchain sans perdre les avantages.
- Frais d'utilisation du smart contract.

Un smart contract pour le recommandé
Un courrier recommandé permet de s'assurer que le document est remis à la bonne personne, ceci grâce à une signature du destinataire. Le cahier des charges décrit plutôt de un historique des envois immuables et vérifiable. Celui-ci permet d'être sûr que la personne A a bien envoyé un document à la personne B, et que la personne B l'a bien reçu.

Dans le cadre de notre projet, un smart contract peut nous servir par exemple à vérifier le format et la structure des données avant de les intégrer dans la blockchain. Cela peut également éviter les transactions en double ou encore simplement nous permettre de retrouver facilement les transactions concernant l'application, car ces dernières passeront toutes par le contrat en question.

De plus, cela simplifie au maximum l'utilisation de la blockchain, les seules transactions réalisées sur celles-ci étant toutes destinées au smart contract.

2.5 Hyperledger

La « Linux Foundation » a pour but de créer un environnement dans lequel des communautés de développeurs et d'entreprises se rencontrent et se coordonnent afin de créer des frameworks de blockchains open-sources. Hyperledger cherche à faire avancer la technologie blockchain en rendant facile son utilisation dans l'industrie. Hyperledger se concentre sur des blockchains non monétaires, c'est pourquoi ces blockchains utilisent des méthodes différentes du minage ou de la preuve d'enjeu afin de vérifier les transactions.

Hyperledger Fabric
Initialement développé par IBM, puis mis à disposition d'Hyperledger, Fabric est un framework de blockchain, permettant à n'importe qui ayant les connaissances nécessaires de mettre en place sa propre blockchain de type Fabric et de l'adapter à ses besoins. Fabric est basé sur une architecture modulaire et supporte des composants « plug-in ». De ce fait ce framework s'adapte à toute sorte de projets et d'industries. Il est par exemple possible de choisir entre plusieurs méthodes de consensus.

Ce type de blockchain supporte également le concept de smart contracts appelé dans ce contexte « chain code ». Les smart contracts permettent de définir comment les données sont stockées dans la blockchain. En effet, Hyperledger n'est pas limité à un système de transactions entre deux comptes afin d'y stocker des données, mais il est possible de stocker les données en les liants à des clés uniques de manière similaire à une base de données.

Les blockchains Hyperledger sont de type consortium, c'est-à-dire qu'elles fonctionnent avec des permissions. Le contenu et l'utilisation de la blockchain n'est pas ouvert à tous.

Hyperledger Sawtooth
Ce framework est développé par Intel et contrairement à Fabric, il permet de mettre en place des blockchains publiques et/ou permissionnées. La méthode de consensus de Sawtooth est basée sur PoET (Proof of Elapsed Time), soit preuve de temp écoulé. Ce système permet une très bonne évolutivité de la taille de la blockchain tout en restant très fluide. Cependant, ce processus nécessite des processeurs Intel spéciaux contenant une instruction spécifique. Sawtooth est plus jeune que Fabric, il y a donc moins de documentation et d'aide disponible à son sujet.

Avantages et inconvénients d'Hyperledger
Avantages

- Contrôle de la blockchain, confidentialité, système de permissions.
- Non-monétaire, donc pas de frais.
- Systèmes de plugins pour s'adapter à différents projets.

Inconvénients

- Pseudo décentralisation.
- Connaissances, temps et infrastructures requises pour la mise en place.

2.6 Hyperledger Composer

Composer est un outil proposé par Hyperledger et compatible avec Fabric. Il permet de facilement « modeler » cette dernière afin de correspondre au mieux à nos besoins. Entre autres, cet outil permet de définir les différents acteurs ayant accès à la blockchain ainsi que leurs droits, mais également quelles données y sont stockées. De plus, Composer offre une API REST permettant d'interagir avec la blockchain.

Composer est un outil idéal pour facilement et rapidement développer des applications proof-of-concept. Cependant pour développer une application pour la production, il est conseillé de développer directement le chaincode pour Fabric.

Pour que Composer puisse déployer une application sur la blockchain, il faut lui fournir une « Business Network définition », c'est un ensemble de fichiers modélisant et décrivant la logique de l'application. Ces fichiers sont ensuite rassemblés dans une archive BNA (Business Network Archive) afin d'être déployée sur le réseau Fabric.

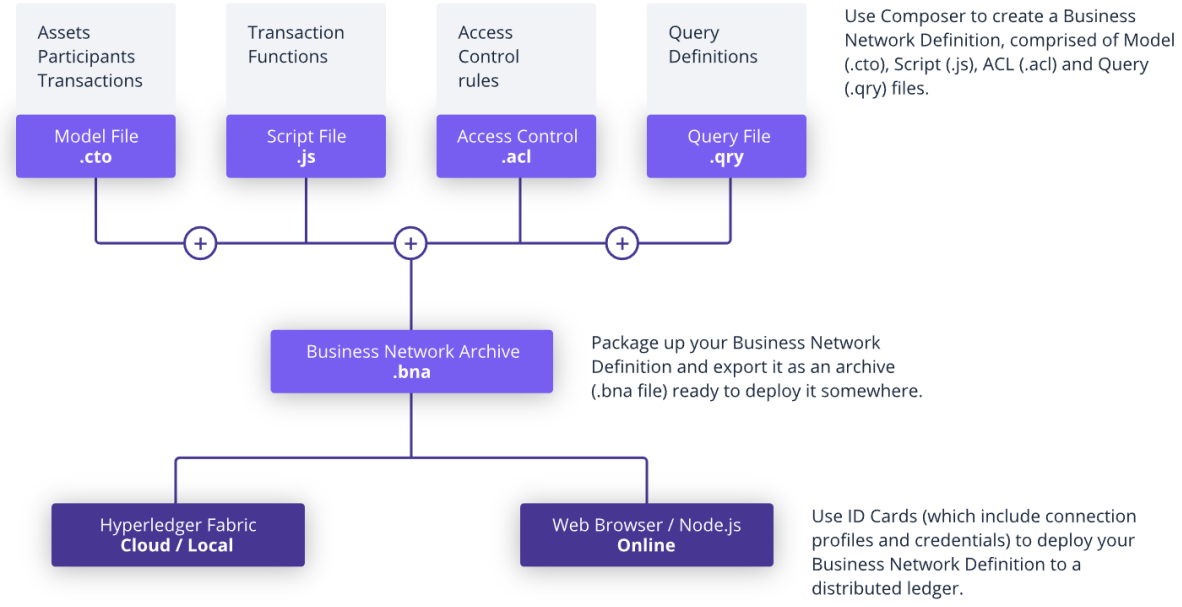


Figure 2 : Schéma du fonctionnement de Composer. Source : [1]

La modélisation du smart contract généré par Composer se fait grâce aux éléments suivants :

Assets	Objet à stocker sur la blockchain composée de propriétés pouvant stocker des informations.
Participants	Utilisateurs capables d'exécuter des transactions et pouvant être liés à des assets.
Transactions	Définition des interactions des participants avec les assets. Logique permettant l'ajout et la modification des assets.

Tableau 3 : Eléments de modélisation de l'outil Composer

Hyperledger Composer propose un outil en ligne appelé « playground » (Documentation [2]) permettant de tester son code de BND sur une blockchain Fabric virtuelle. Cet outil indique également si il y a des erreur de syntaxes dans les fichiers du modèle.

L'équipe de développement de Composer a annoncé en fin 2018 la fin du développement de cet outil. En effet ils ne vont plus ajoutés de nouvelles fonctionnalités. L'architecture devenant de plus en plus complexe à maintenir, ils ont fait le choix de plutôt facilité le développement natif sur Hyperledger Fabric (source : [3]).

2.7 Solutions

Plusieurs solutions différentes peuvent être mises en place pour arriver à un résultat semblable. Nous avons détaillé les solutions envisageables dans ce sous-chapitre.

2.7.1 Utilisation d'une blockchain publique ou privée

L'utilisation d'une blockchain publique telle que Ethereum offre des avantages, comme une bonne décentralisation et une infrastructure déjà en place. Par contre, la sécurisation des données y est plus difficile étant donné que celles-ci sont publiques et accessibles à n'importe qui à l'aide d'un simple explorateur blockchain. Il faut également payer des frais pour chaque transaction et pour l'utilisation de smart contracts.

Une blockchain privée de type Hyperledger Fabric permet de contrôler les accès et ainsi de garder le contrôle sur les données. Ce framework permet également de modéliser la blockchain pour qu'elle corresponde à nos besoins spécifiques et ne nécessite pas de frais d'utilisation. Il est cependant nécessaire de mettre en place et maintenir toute l'infrastructure, dont plusieurs nœuds à repartir au sein de différentes organisations afin de garantir la décentralisation des données.

2.7.2 Multicomptes sur la blockchain

Une des possibilités pour la réalisation de ce projet est de représenter chaque utilisateur final de notre application (envoyeur et receveur) directement sur la blockchain. Dans ce cas de figure, chaque utilisateur disposerait d'un compte personnel (clé publique et privée) sur la blockchain. C'est ainsi que fonctionnent la plupart des dApps. Sans avoir recours à un smart contract, les utilisateurs pourraient s'échanger les IDs représentants les documents directement entre eux en utilisant le champ « data » des transactions. Notre application s'occuperait de réaliser ces transactions et de faire le lien entre les utilisateurs de la plateforme Web et leurs comptes sur la blockchain, de manière à rendre l'utilisation plus aisée pour une personne lambda. Ainsi, les clés publiques et privées seraient stockées dans une base de données, et les utilisateurs n'en auraient pas connaissance.

La seconde possibilité est d'utiliser un seul compte sur la blockchain afin de réaliser les interactions entre notre application et un smart contract (ou un autre compte). Pour chaque envoi de document, on crée une transaction contenant les informations de l'expéditeur, du destinataire, ainsi que l'ID du document et la date.

Cette deuxième possibilité empêche des analyses statistiques des comptes effectuant les transactions et simplifie grandement l'implémentation.

2.8 Notre solution

Afin de correspondre au mieux aux attentes du mandant, nous avons choisi de nous diriger vers une blockchain privée en utilisant le framework Hyperledger Fabric. En effet, cette solution est bien plus adaptée à une utilisation en entreprise car elle remplit les attentes au niveau de la protection des données. De plus, cela permet de mettre en place facilement plusieurs nœuds à différents emplacements, ce qui est particulièrement adapté à notre cas. En effet, cette plateforme étant destinée à être utilisé par plusieurs entités étatiques, chacune pourrait disposer de son propre nœud afin d'assurer les transactions.
Hyperledger Composer ajoute une couche d'abstraction au-dessus de la blockchain, nous permettant de développer des smart contracts de manière simplifiée en utilisant le langage Javascript. De plus, il génère directement une API REST que nous allons pouvoir utiliser pour interagir avec la blockchain.

Nous n'allons utiliser qu'un seul compte blockchain pour effectuer les transactions étant donné qu'il y a peu d'avantages à en utiliser plusieurs dans le cadre de notre projet. De plus, les utilisateurs étant en réalité déjà géré d'une manière externe (par exemple à l'aide d'LDAP), il est moins complexe dans un premier temps de les garder à l'écart de la blockchain, même si Hyperledger permet l'intégration de LDAP.

3 Spécifications

Ce chapitre décrit la phase de conceptualisation qui a précédé la réalisation du projet.

3.1 Fonctionnement global

Notre proof-of-concept sera donc composé d'une application WEB qui interagira avec la blockchain via une API REST.
Cette application permet de mettre en œuvre une authentification simple afin d'identifier les utilisateurs grâce à une base de données. L'interface Web sera semblable à une boite mail et permettra aux utilisateurs d'envoyer et de recevoir des documents (représentés par des Document IDs).

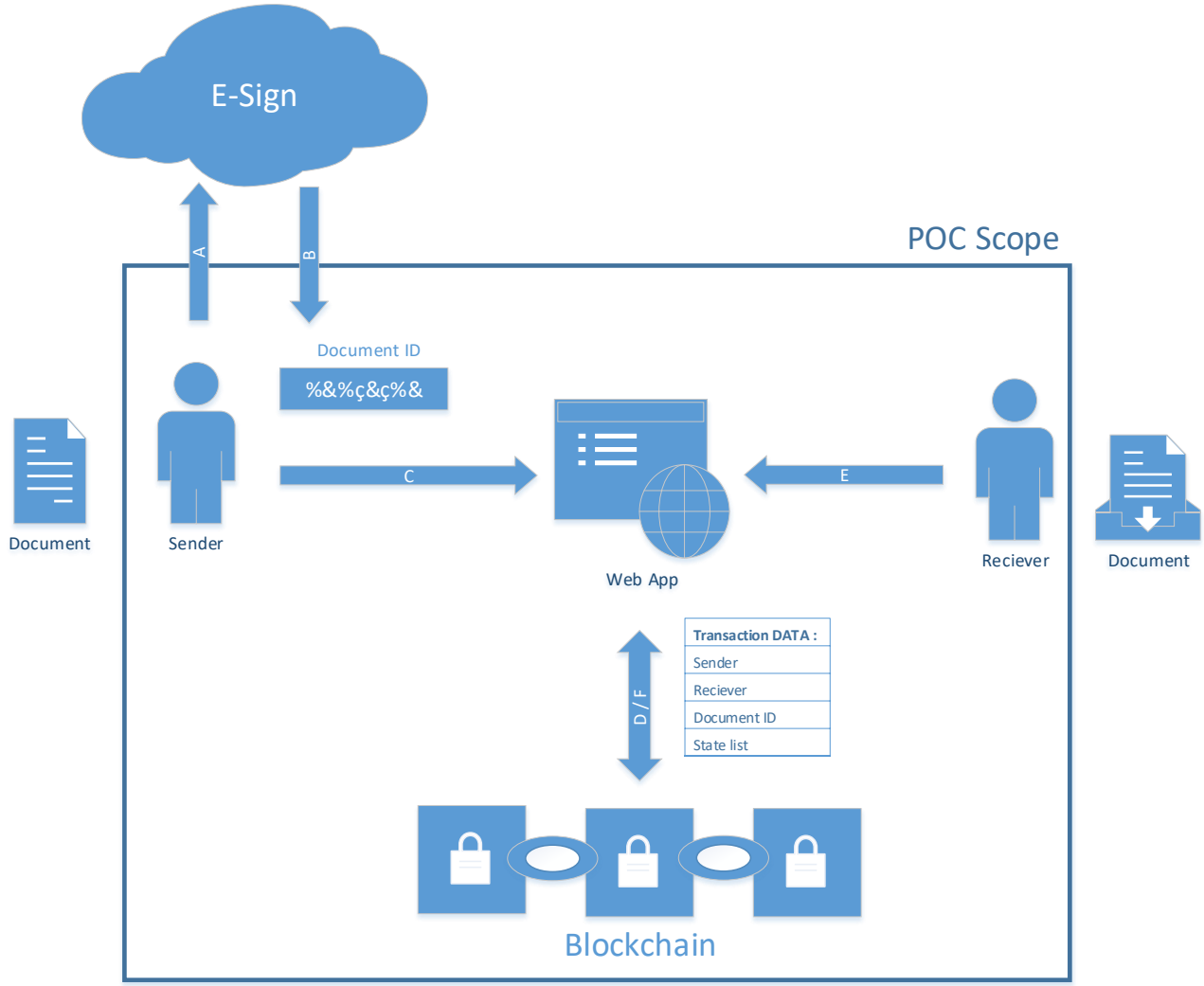


Figure 3 : Schéma du fonctionnement global du proof-of-concept

A – L'utilisateur upload son document sur la plateforme E-Sign
B – E-Sign génère un URL contenant un ID donnant accès au document signé.
C – L'utilisateur se connecte sur l'application Web et envoie l'ID au destinataire.
D – La transaction concernant cet envoi est stocké dans la blockchain.
E – Le destinataire se connecte et constate qu'il a reçu l'ID du document
F – L'état du document est modifié dans la blockchain

3.1.1 Limites de notre Proof-of-Concept

Dans le cadre de notre prototype, nous ne nous intéresserons pas à la gestion du document en lui-même, celui-ci étant hébergé sur une plateforme externe (E-Sign). Nous allons travailler uniquement avec des IDs les représentant. Nous ne gérons pas de lien direct avec E-Sign, c'est à l'utilisateur d'entrer par lui-même l'ID du document pour l'envoyer.

3.1.2 Etats de l'envoi

Afin de pouvoir suivre l'état de l'envoi d'un document, celui-ci est décrit dans un tableau stocké comme attribut du document. Chaque statut est composé d'un titre (SENT, RECEIVED et READ expliqué ci-dessous) ainsi que de la date à laquelle il a été défini.

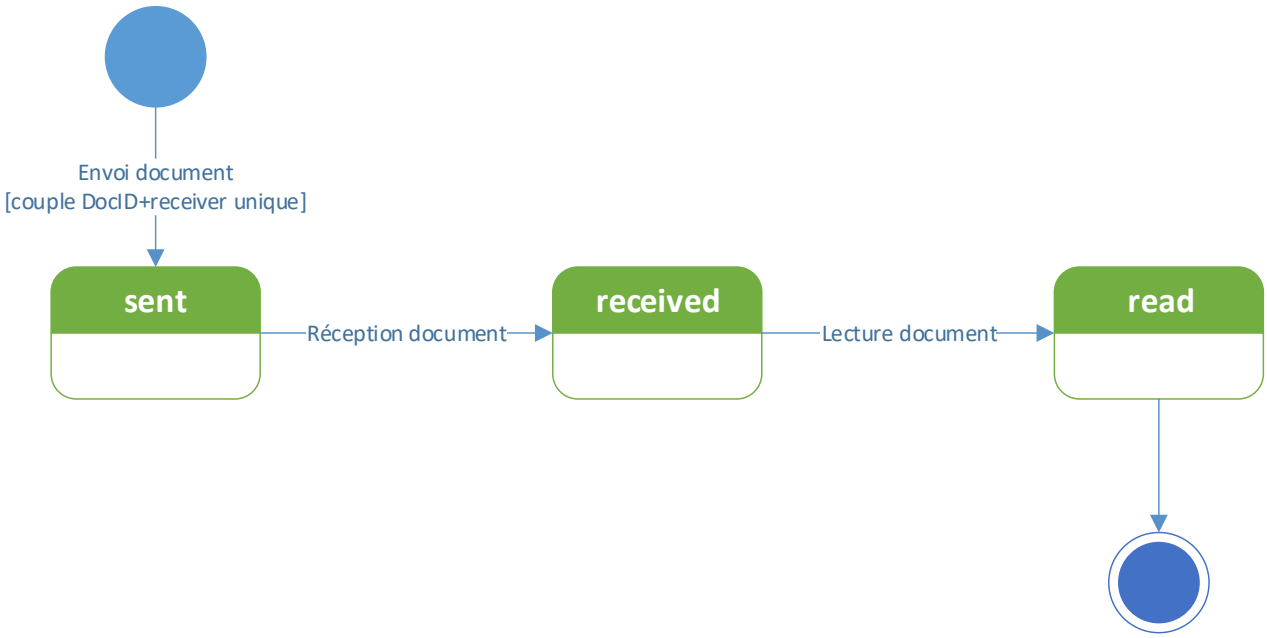


Figure 4 : Diagramme des états d'un envoi

Un document va passer par trois statuts qui vont être ajoutés dans l'ordre suivant :

Étape	Nom	Définition	Évènement
1	SENT	Le document a été envoyé	L'objet a été créé sur la blockchain
2	RECEIVED	Le document a été reçu	Le destinataire s'est connecté à la plate-forme (récupération de ses documents reçus)
3	READ	Le document a été lu	Le destinataire a cliqué sur le document depuis la plateforme

Tableau 4 : Etats possible d'un document

3.2 Use Case

Nous avons défini quatre cas d'utilisations pour notre application : Authenticate (connexion d'un utilisateur), Logout (déconnection d'un utilisateur), Send Document ID (envoi d'un document) et Check received IDs (réception des documents pour un utilisateur donné).

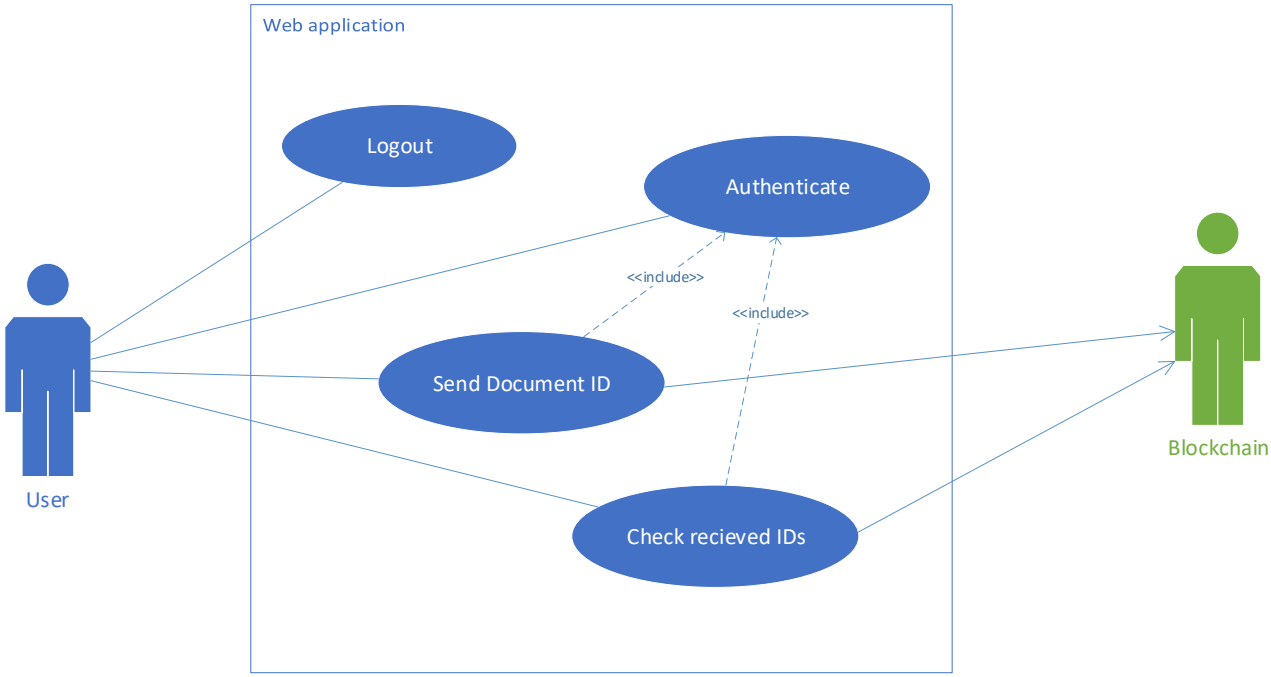


Figure 5 : Diagramme de cas d'utilisations

Les différents use cases sont décrits dans les fiches descriptives en annexe.

3.3 Diagramme des composants

Ce diagramme représente les différentes parties de l'environnement de notre application et leurs méthodes d'interactions.

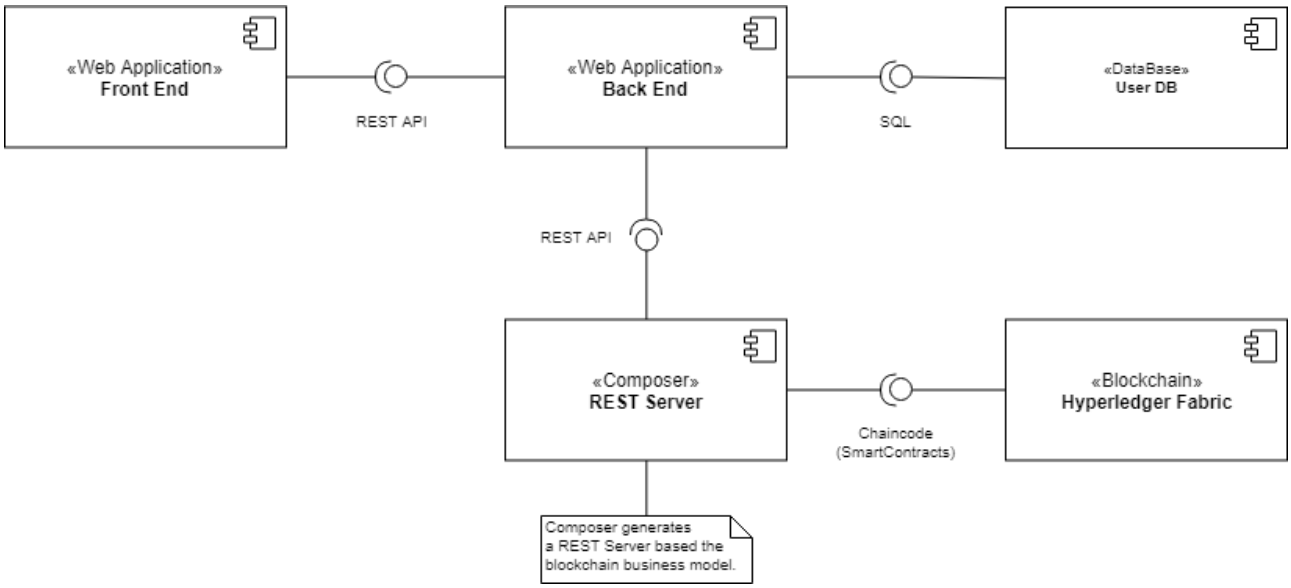


Figure 6 : Diagramme des composants

On peut ainsi voir que notre front-end va communiquer avec le back-end à l'aide d'une API REST. Le back-end va interagir avec deux éléments distincts : la base de données SQL pour la gestion des utilisateurs et le serveur REST d'Hyperledger Composer pour l'envoi et la réception de document. Ce dernier est en lien direct avec la blockchain Hyperledger Fabric. Ils communiquent ensemble grâce au SDK.

3.4 Blockchain

3.4.1 Données stockées

Le stockage des données sur la blockchain se fait sous forme d'objets. Voici les différents champs que nous allons implémenter :

Transaction Object	
UID	Identifiant permettant de représenter un document de manière unique.
Events	Tableau représentant les états. Chaque entrée contient l'état en question avec un timestamp.
Sender	ID représentant un utilisateur de la base de données.
Receiver	ID représentant un utilisateur de la base de données.
Document ID	ID représentant un document stocké sur la plateforme E-Sign.

Tableau 5 : Données contenu dans une transaction

L'identificateur unique d'un objet stocké sera le couple « Document ID » et « Receiver ».
Il sera possible d'envoyer un document à de multiples destinataire depuis notre application Web. Ceci génèrera pour chaque destinataire un objet différent sur la blockchain.

3.4.2 Smart contracts

Nous allons utiliser l'outil Composer afin de générer le smart contract qui va être utilisé pour interagir avec la blockchain.

Les fonctions qui devront être mises en place dans le smart contract sont les suivantes :

- Envoi d'un document sur la blockchain.
- Mise à jour du statut d'un document (tableau d'événements).
- Accès à la liste des documents envoyés et reçus par un utilisateur spécifique.
- Interdire l'envoi à double d'un document à une même personne.

3.5 Back-End

Le back-end sera réalisé en ASP.NET Core afin de répondre aux demandes du mandant.

Celui-ci sera composé de deux parties distinctes :

- L'authentification des utilisateurs à l'aide d'une base de données SQLite
- Les interactions avec la blockchain via l'API REST Composer

L'application Web va pouvoir communiquer avec le back-end grâce à une API REST. Celui-ci va lui renvoyer des réponses au format JSON.

3.6 Front-End

Le front-end sera réalisé en Angular afin de répondre aux demandes du client.

Un framework Javascript est idéal pour ce type de projet étant donné qu'il permet de développer des applications Web fluides pour le visiteur. En effet, les ressources de l'application sont mises en cache en local sur la machine cliente et des appels API sont passés au back-end lorsque nécessaire. On peut ainsi développer une application capable de fonctionner dans de mauvaises conditions (coupure réseau, débit faible), ce qui convient typiquement à un usage mobile.

Angular est un framework Javascript développé depuis 2016 par Google. Il repose sur la structure MVVM (Modèle Vue Vue-Modèle). La logique de l'application est développée en Typescript, tandis que la vue est un mélange d'HTML et de langage template permettant d'interagir avec le modèle de vue.

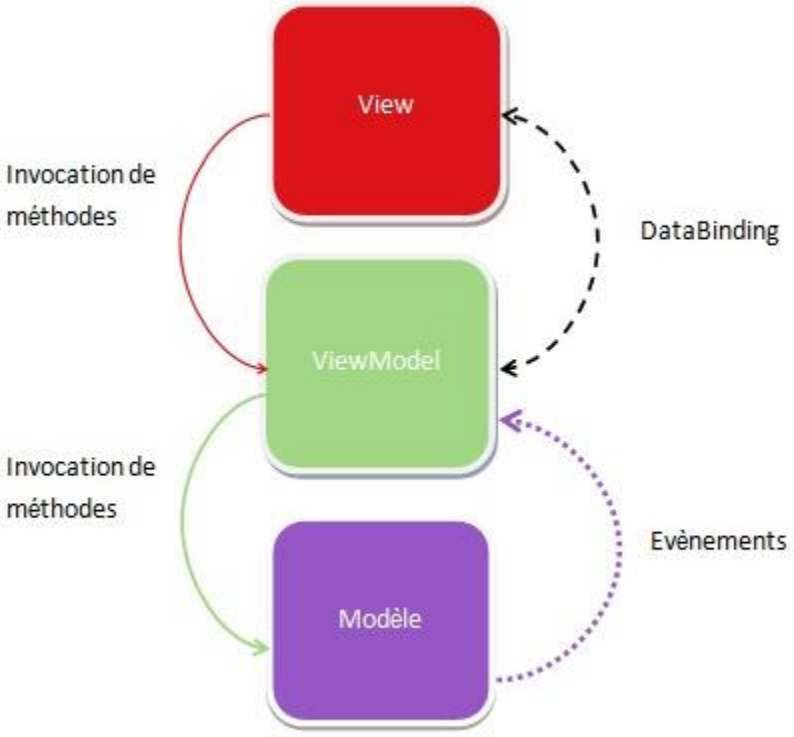


Figure 7 Diagramme MVVM. Source : [4]

4 Réalisation

Dans ce chapitre, nous expliquons concrètement ce que nous avons mis en place pour créer notre prototype ainsi que les points à améliorer et les problèmes rencontrés.

4.1 Infrastructure Blockchain

L'infrastructure d'une blockchain Hyperledger demande beaucoup de temps pour être installée. Nous avons décidé de nous concentrer sur l'aspect du développent des smart contracts grâce à l'outil Composer. Nous avons ainsi bénéficié de l'aide de Cyril Vallélian, collaborateur de l'école, qui nous a fourni l'infrastructure Hyperledger et Composer. Cette infrastructure a été mise en place sur une machine virtuelle du réseau de l'école.

4.2 Hyperledger Composer

Le code Composer se divise en différents fichiers remplissant chacun un rôle distinct.

Voici la structure des fichiers du BND (Business Network Definition). Ces différents fichiers contiennent la logique utilisée par Composer afin de générer le chaincode utilisé par la blockchain.

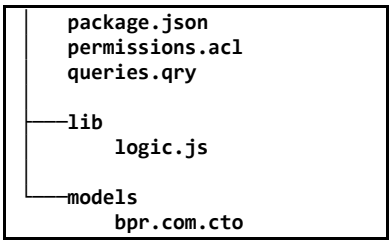


Figure 8 : Structure des fichiers du BND

package.json	Contient les métadonnées du business network.
permissions.acl	Définition des droits d'accès aux assets et aux fonctions.
queries.qry	Définitions des requêtes permettant d'accéder à l'état le plus récent des données stockées dans la blockchain.
logic.js	Fonctions en Javascript associées aux transactions. Logique de chaque transaction.
bpr.com.cto	Contient la modélisation des éléments stockés sur la blockchain : assets, transactions et participants. Utilise un langage propre à Composer (documentation : [5]).

Tableau 6 : Fichiers du BND

4.2.1 Modélisation

Le diagramme de classe suivant représente la structure sous laquelle les données sont stockées.

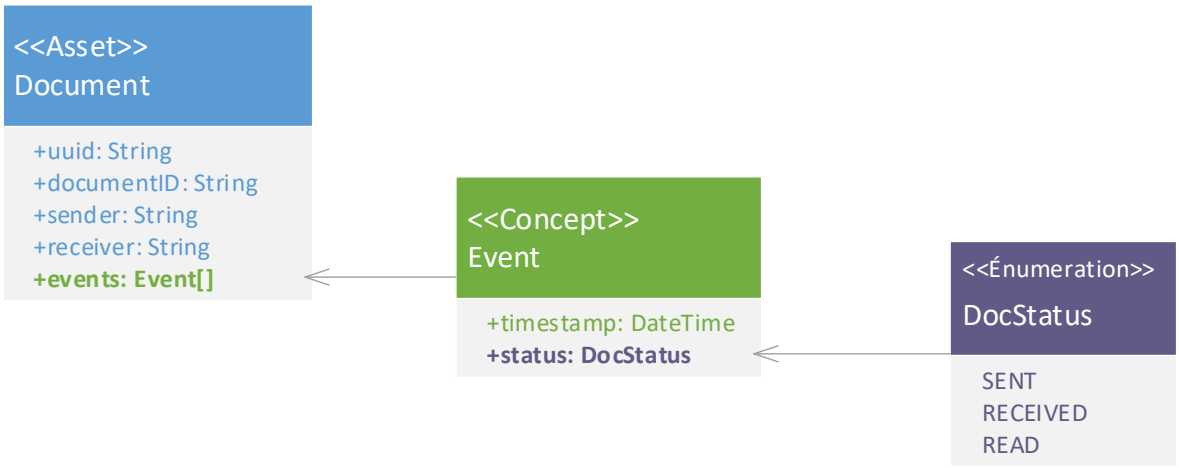


Figure 9 : Structure des données définie dans le fichier bpr.com.cto

Voici le code permettant de modéliser les documents dans le fichier bpr.com.cto :

```
// ASSET representing the a document sent on the blockchain.
asset Document identified by uuid {
  o String uuid              // Unique identifier used by hyperledger, constructed by documentID_receiver.
  o String documentID        // Stores the document ID/HASH representing the doc on E-Sign.
  o String sender            // User ID of the user who sent the document.
  o String receiver          // User ID of the receiver of the document.
  o Event[] events           // Array containing the document events (sent, received, read).
}
```

Le diagramme suivant représente des transactions sur la blockchain. Ce sont les méthodes permettant d'interagir avec les données stockées sur la blockchain (dans notre cas : l'asset Document).

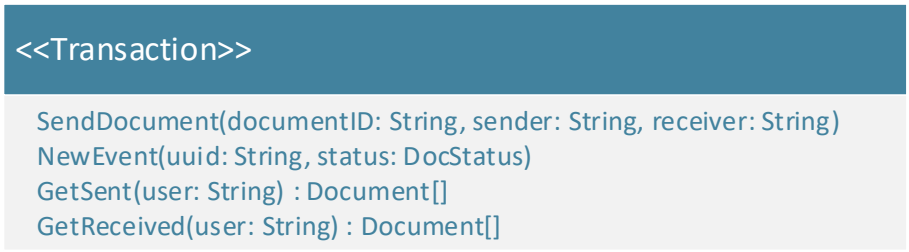


Figure 10 : Transactions définie dans le fichier bpr.com.cto

4.2.2 Transactions

Pour chaque transaction définie dans le modèle (fichier bpr.com.cto), nous avons défini une logique en Javascript.

Voici le code permettant de modéliser la transaction d'envoi d'un document (fichier bpr.com.cto) :

```
// Transaction used to add a new document on the blockchain
transaction SendDocument {
  o String documentID        // Parameter : Stores the document ID/HASH representing the doc on E-Sign.
  o String sender            // Parameter : User ID of the user who sent the document.
  o String receiver          // Parameter : User ID of the receiver of the document.
}
```

Et la logique correspondante (fichier logic.js) :

```
/**
 * Create a new Document asset on the blockchain.
 * @param {bpr.com.SendDocument} params
 * @transaction
 */
async function sendDocument(params) {
     const factory = getFactory();
     var uuid = params.documentID + '_' + params.receiver; // Creating a unique identifier using doc ID
and receiver.

     const document = factory.newResource(namespace, 'Document', uuid); // Creating a new document asset.
     var createTimestamp = new Date(); // Get the curent date time for the SENT event.
     document.documentID = params.documentID;
     document.sender = params.sender;
     document.receiver = params.receiver;

     var event = factory.newConcept(namespace, "Event"); // Creating the event structure.
     event.timestamp = createTimestamp;
     event.status = 'SENT';
     document.events = [event]; // Store the SENT event in the document.

     // Save the document on the blockchain.
     const assetRegistry = await getAssetRegistry(document.getFullyQualifiedType());
     await assetRegistry.add(document);

     // Emit event
     const evt = factory.newEvent(namespace, 'SendDocumentEvent');
     evt.doc = document;
     emit(evt);
}
```

Les transactions peuvent retourner des données lors de leur appel. Dans notre cas, « getSent » et « getReceived » retournent une liste de documents. Pour cette fonction, nous utilisons le modificateur « @returns(Document[]) » lors de la modélisation :

```
// Transaction used to get sent documents of the specified user.
// Commit(false) means the transaction won't be added to the blockchain, because we are only reading data
from it.
// Returns a list of documents.
@commit(false)
@returns(Document[])
transaction GetSent {
  o String user              // Parameter : User UID.
}
```

Le modificateur « @commit(false) » indique que cette transaction ne modifie pas la block-chain ; elle ne fait que de retourner des données stockées.

La transaction « GetReceived » a comme spécificité de directement ajouter l'événement « RECEIVED » aux documents lors de son appel. Ceci évite des appels API pour ajouter un évènement à chaque document chargé. Cela assure également le fait que ce statut sera obligatoirement ajouté lors de la réception de documents, étant donné qu'il est réalisé au plus haut niveau (blockchain).

```
/**
 * Get received documents for specified user using 'selectReceiverDocuments' query form .qry file.
 * @param {bpr.com.GetReceived} params
 * @returns {bpr.com.Document[]} Array of documents
 * @transaction
 */
async function GetReceived(params){
    // Call the query.
    let assetRegistry = await getAssetRegistry('bpr.com.Document');
    let results = await query('selectReceiverDocuments', { user: params.user });

    // Modify each document to add the the RECEIVED event.
            const factory = getFactory();
            var i;
            for (i = 0; i < results.length; i++) {
                    var document = results[i];
                    if(document.events.length == 1){
                            var event = factory.newConcept(namespace, "Event");
                            event.timestamp = new Date();
                            event.status = 'RECEIVED';
                            document.events.push(event);
                            assetRegistry.update(document);
                    }
            }
    return results;
}
```

4.2.3 Requêtes

Les requêtes Composer permettent de récupérer des données directement depuis le « world state ». Le « world state » est une base de données CouchDB contenant le dernier état des éléments stockés sur la blockchain. Celle-ci est mise à jour de manière automatique et transparente après chaque ajout/modification de données dans la blockchain. Elle permet d'accélérer les requêtes en évitant de devoir questionner toute la blockchain afin d'y retrouver des données.

Nous utilisons cette fonctionnalité dans les transactions « GetSent » et « GetReceived » afin de récupérer les documents concernant un utilisateur spécifique.

Voici la requête permettant de récupérer les documents envoyés par un certain utilisateur. Fichier queries.qry :

```
query selectSenderDocuments {
  description: "Select documents where sender is particular user"
  statement:
      SELECT bpr.com.Document
          WHERE (sender==_$user)
}
```

Et voici l'appel de cette requête depuis la fonction « GetSent » du fichier logic.js :

```
/**
 * Get sent documents for specified user using 'selectSenderDocuments' query from .qry file.
 * @param {bpr.com.GetSent} params
 * @returns {bpr.com.Document[]} Array of documents
 * @transaction
 */
async function GetSent(params){
    let assetRegistry = await getAssetRegistry('bpr.com.Document');
    let results = await query('selectSenderDocuments', { user: params.user });
    return results;
}
```

4.2.4 Évènements

Composer propose un système d'évènements permettant à des applications externes de s'y abonner grâce à un appel d'API. Ces évènements doivent être définis dans le fichier bpr.com.cto. Ils peuvent ensuite être émis dans une fonction de transaction contenue dans le fichier logic.js.

Nous avons défini deux événements (documentation : [6]) qui ne sont pas utilisés actuellement par notre proof-of-concept. Le premier permet de signaler un nouvel ajout de document (donc un nouvel envoi). Le second concerne l'ajout d'un événement (ex : READ) à un document existant. Ces événements permettraient au front-end de se tenir à jour sans faire de polling.

Voici la définition de l'événement d'envoi d'un nouveau document (bpr.com.cto) :

```
// Event emmited after adding a document. Not currrently used.
event SendDocumentEvent {
  --> Document doc   // The newly added document.
}
```

Puis son émission à la fin de la fonction d'envoi d'un document (logic.js) :

```
// Emit event
const evt = factory.newEvent(namespace, 'SendDocumentEvent');
evt.doc = document;
emit(evt);
```

Nous n'avons pas implémenté cette fonctionnalité dans le reste de l'application car elle aurait demandé trop de temps. Notre architecture avec le back-end REST au centre ne permet pas de transmettre des événements directement au front end. Cependant, cela aurait été possible en utilisant une communication en direct entre le front-end et back-end, par socket par exemple.

4.2.5 API REST

Composer fournit un outil permettant de générer une API en fonction de notre modélisation. Dans notre cas, le script d'initialisation de la blockchain fourni par Cyril Vallélian lance directement la génération de l'interface REST.

Il est possible de modifier cette API afin de correspondre aux besoins spécifiques et d'y réaliser un contrôle d'accès par authentification (documentation [7]).

Nous avons décidé de garder la configuration par défaut de l'API et de ne pas y ajouter d'authentification, ceci pour deux raisons :

Premièrement dans un cas de production, il serait plus adapté de réaliser le smart contract dans le langage natif utilisé par Hyperledger Fabric. En effet, Composer est un outil qui facilite grandement le développement d'un smart contract (intéressant pour un proof-of-concept), mais il n'est plus maintenu. Ce point est expliqué plus en détail dans le chapitre « Futur et améliorations ».

Deuxièmement, le serveur back-end est situé sur la même machine que l'API REST de Composer. Ainsi, les communications se font en local uniquement.

L'accès à cette API est disponible par le port 8080.

a) Appels

Ce schéma représente les actions disponibles pour cette API en spécifiant quel format JSON doit être joint pour les requêtes.

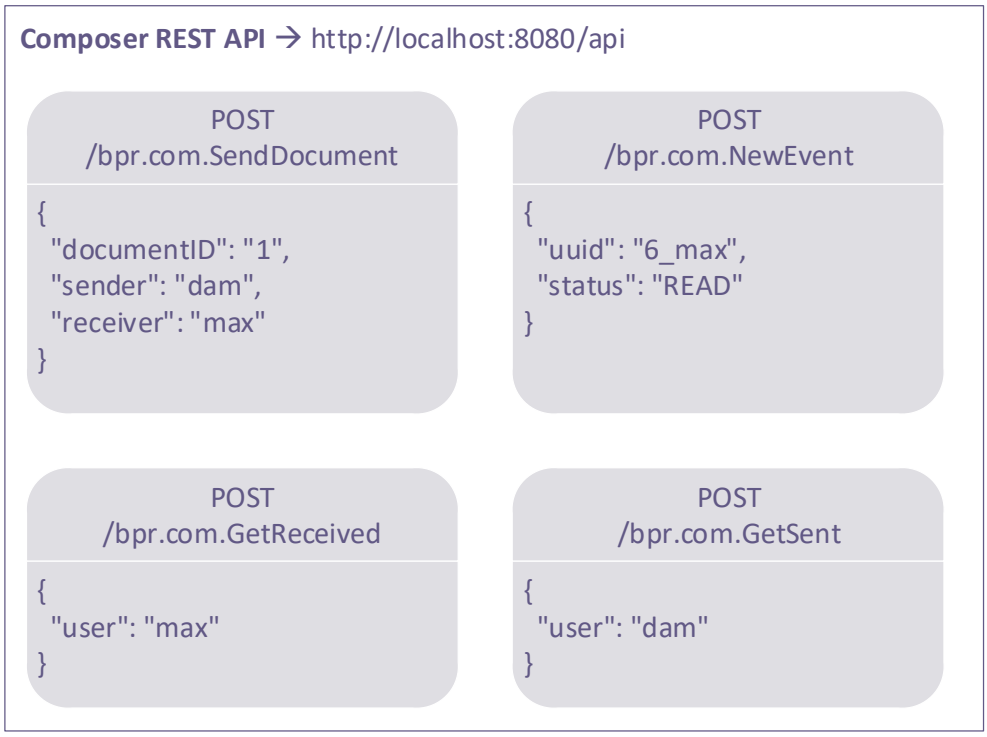


Figure 11 : Schéma des actions REST de Composer

b) Retours

Les données JSON retournées après l'appel de ces actions sont les suivantes :

GetSent et GetReceived

Ces deux actions renvoient les données de la blockchain sous forme d'un tableau de documents.


```
[
    {
        "$class": "bpr.com.Document",
        "uuid": "Hello_92af7561-8e3b-40a7-9388-fde5f93a9f28",
        "documentID": "Hello",
        "sender": "2498ba7d-0ba9-4460-b767-9c12b09e431c",
        "receiver": "92af7561-8e3b-40a7-9388-fde5f93a9f28",
        "timestamp": "2019-01-24T15:03:21.481Z",
        "events": [
            {
                "$class": "bpr.com.Event",
                "timestamp": "2019-01-24T15:03:21.481Z",
                "status": "SENT"
            },
            {
                "$class": "bpr.com.Event",
                "timestamp": "2019-01-24T15:05:35.555Z",
                "status": "RECEIVED"
            }
        ]
    }
]
```


SendDocument et NewEvent

Ces actions ne retournent pas de données spécifiques. Cependant, Composer renvoie les informations sur la transaction qui a été effectuée.


```
{
    "$class": "bpr.com.SendDocument",
    "documentID": "1",
    "sender": "dam",
    "receiver": "max",
    "transactionId": "761db5373dba5e307d17b1ac99799c177f4451f57564203a28e919e519346754"
}
```


Erreurs

Si une transaction appelée génère une erreur (par exemple s'il manque des données essentielles dans l'appel), Composer retourne une erreur au format suivant. A noter que même en cas d'erreur, le statut HTTP de la réponse reste « 200 OK ».


```
{
    "error": {
        "statusCode": 400,
        "name": "SyntaxError",
        "message": "Unexpected token \r in JSON at position 14",
        "body": "{\r\n  \"user\": \"\r\n}",
        "status": 400,
        "stack": "SyntaxError: Unexpected token \r in JSON at position 14\n ... "
    }
}
```

4.3 Back-End

Notre back-end est un serveur REST réalisé en .NET Core version 2.2. Il sert d'intermédiaire entre Composer et le front-end. Il s'occupe principalement de l'authentification des utilisateurs de l'application.

Les technologies et outils utilisés sont :

- .NET Core v2.2 Web API
- Authentification avec Identity Service
- Base de données SQLite
- Authorization JWT (JSON Web Token)

Voici le diagramme de classe du back-end :

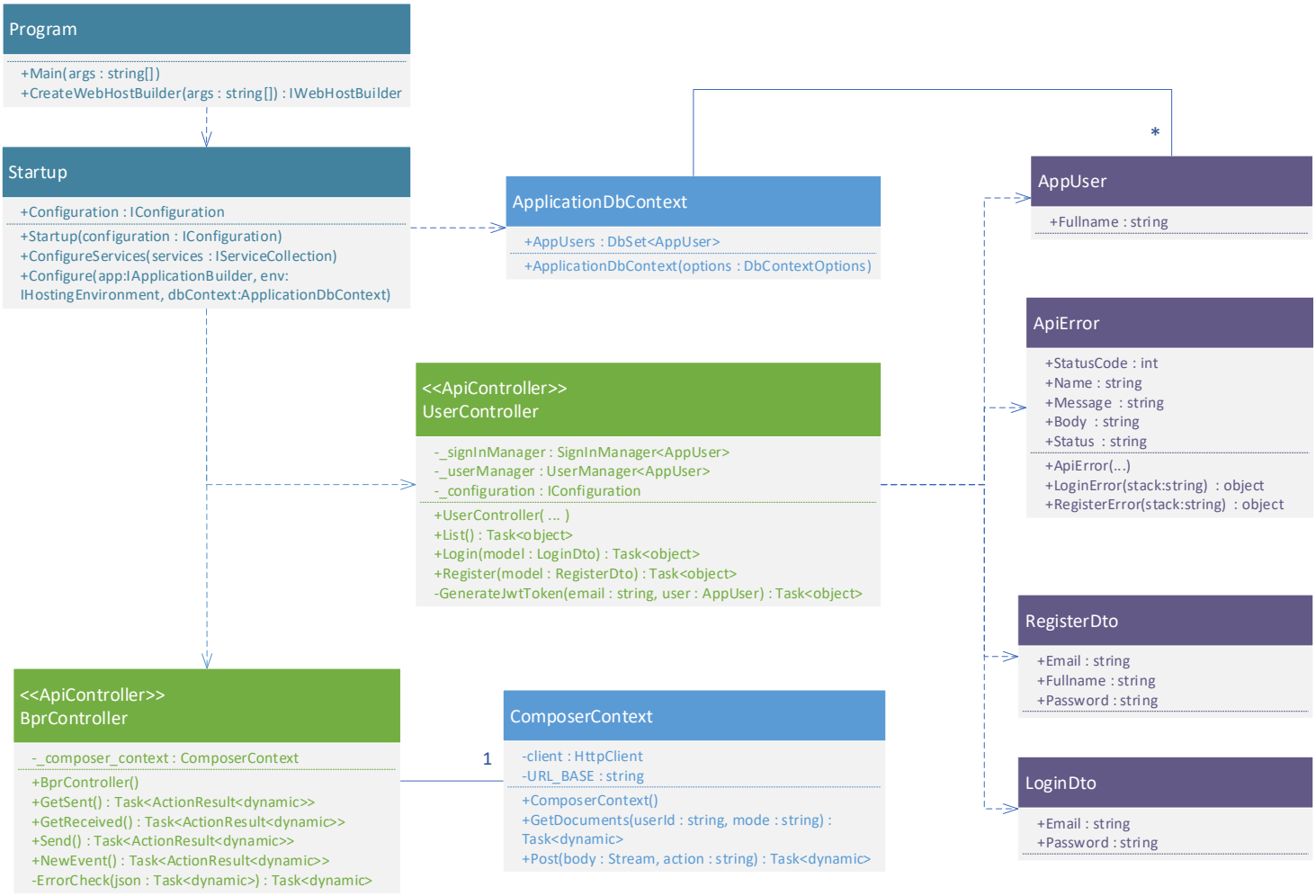


Figure 12 : Diagramme de classe du back-end

L'accès à l'API du back-end est disponible par le port 5001.

4.3.1 Authentification des utilisateurs

Nous utilisons le système d'identités intégré à .NET Core. Cette fonctionnalité s'occupe de stocker les données des utilisateurs dans une base de données. Nous avons choisi d'utiliser SQLite pour plus de facilité. La base de données est stockée dans le fichier « appDB.db » à la racine du projet .NET.

Les informations des utilisateurs sont les suivantes :

Propriété	Utilité
uuid	Identifiant unique
fullname	Prénom et nom
email	Adresse e-mail
password	Mot de passe

Tableau 7 : Propriétés utilisées pour décrire un utilisateur

Les champs uuid, email et password font déjà partie de la classe « IdentityUser » de .NET. Afin d'ajouter le champ fullname, il a fallu créer la classe « AppUser » héritant de la classe « IdentityUser ».

Les sessions sur le front-end et l'autorisation d'appels à l'API sont gérée grâce à des JWT (JSON Web Token). Le back-end génère le token lors du login et le transmet au front-end. Ce dernier doit incluse le JWT dans le header des requêtes.

Le point d'entrée de l'API concernant les utilisateurs est géré par la classe contrôleur « UserController ».

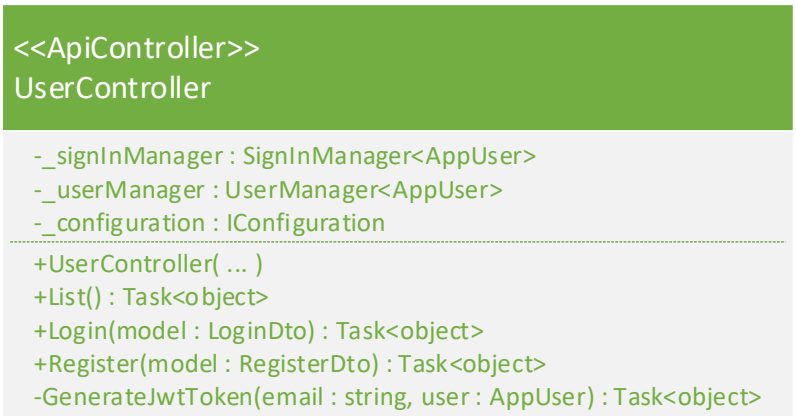


Figure 13 : Classe du contrôleur "UserController"

La méthode « List » sert à fournir la liste de tous les utilisateurs de l'application afin que le front-end puisse y avoir accès (utilisé dans le formulaire d'envoi : liste déroulante des destinataires). Il faut être authentifié pour appeler cette méthode.

Les méthodes « login » et « register » utilisent les classes modèles « LoginDto » et « RegisterDto ». Ces classes servent à stocker les données reçues par le front-end pour y avoir accès facilement. Elles sont déclarées dans le même fichier que la classe « UserController ».
La méthode « Register » n'est pas utilisée par le front-end, car nous n'avons pas eu le temps de d'implémenter cette fonctionnalité. Nous conseillons d'utiliser le logiciel « Postman » pour appeler l'action « Register » et ajouter un nouvel utilisateur.

Le schéma suivant représente les différentes actions possibles ainsi que le format JSON requis pour les requêtes POST.

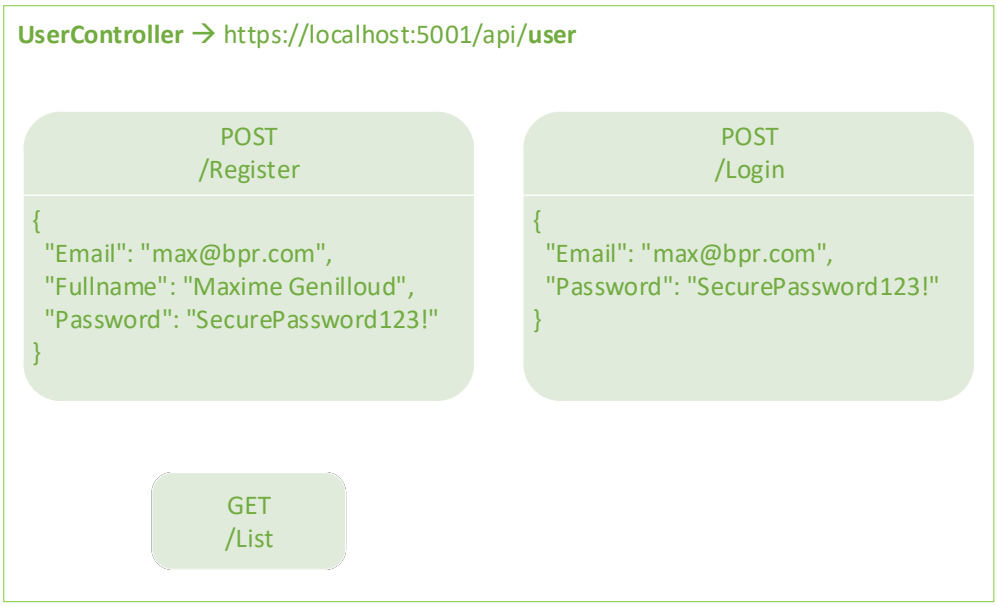


Figure 14 : Schéma des actions REST de UserController

Les données qui sont retournées utilisent les formats JSON suivants :

List
Retourne la liste des utilisateurs de l'application.

```
[
    {
        "uuid": "92af7561-8e3b-40a7-9388-fde5f93a9f28",
        "fullname": "Damien Morattel",
        "email": "dam@bpr.com"
    },
    {
        "uuid": "2498ba7d-0ba9-4460-b767-9c12b09e431c",
        "fullname": "Maxime Genilloud",
        "email": "max@bpr.com"
    }
]
```

Login et Register
Retournent les informations sur l'utilisateur et le JWT permettant l'accès aux méthodes protégées.


```
{
    "email": "dam@bpr.com",
    "uuid": "92af7561-8e3b-40a7-9388-fde5f93a9f28",
    "fullname": "Damien Morattel",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkYW1AYnByLmNvbSIsImp0aSI6Ijg0YzMzZTkwLWE0OTgtNDUwNy1hNTQ2LWYyM2UzZGY3OGZhYSIsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL25hbWVpZGVudGlmaWVyIjoiOTJhZjc1NjEtOGUzYi00MGE3LTkzODgtZmRlNWY5M2E5ZjI4IiwiZXhwIjoxNTUxMTkyNzI5LCJpc3MiOiJodHRwczovL2xvY2FsaG9zdDo1MDAxIiwiYXVkIjoiaHR0cHM6Ly9sb2NhbGhvc3Q6NTAwMSJ9.Qps0A8bYLWW3lfp8lzpP8wXieSEuTQyfzjF3P-VGnCY"
}
```


4.3.2 Interactions avec la blockchain

Le point d'entrée de l'API permettant l'accès à Composer est géré par la classe contrôleur « BprController ».

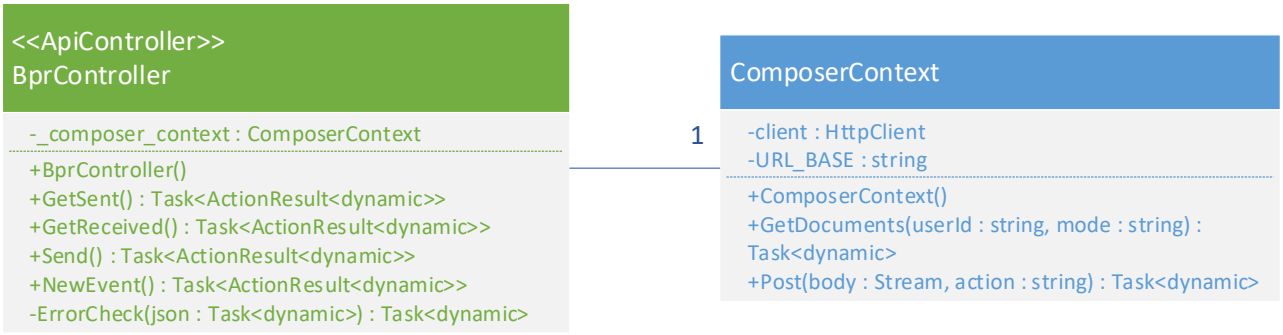


Figure 15 : Classes utilisées pour les interactions avec Composer

La classe « ComposerContext » est utilisée pour communiquer avec le serveur REST de Composer. Nous effectuons les requêtes vers Composer grâce à un client HTTP de type « HttpClient ».

Pour les actions « GetSent » et « GetReceived », le contrôleur se charge de déterminer l'UID de l'utilisateur courant. Pour les actions « Send » et « NewEvent » il retransmet simplement les données de la requête reçue du front-end.

Le contrôleur se charge également de contrôler si la réponse de Composer est une erreur. Dans ce cas, on change le « status code » de la réponse HTTP pour signaler l'erreur au front-end.

Les méthodes de ce contrôleur sont protégées pour qu'uniquement un utilisateur connecté puisse les appeler. On vérifie cela grâce aux JWT transmis lors des requêtes du front-end.

Le schéma suivant représente les différentes actions possibles ainsi que le format JSON requis pour les requêtes POST concernant ce contrôleur.

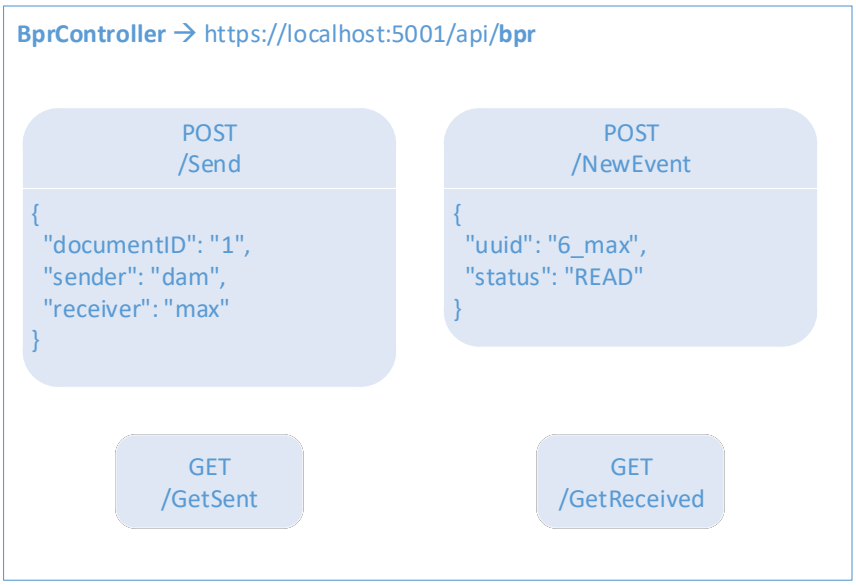


Figure 16 : Schéma des actions REST de BprController

Les réponses renvoyées vers le front-end sont au même format JSON que lorsque l'on contact directement le serveur REST de Composer. En effet, les réponses sont simplement retransmises sans être modifiées par le back-end.

4.3.3 Gestions des erreurs

Comme vu précédemment, les erreurs survenant sur la blockchain sont directement relayées au front-end sans modification. Pour faciliter la gestion des erreurs par le front-end, nous avons décidé d'utiliser le même format d'erreur JSON pour les problèmes survenant avec la partie gestion d'utilisateur (login et register).

Pour ce faire, la classe « ApiError » a été créée avec toutes les propriétés nécessaires.

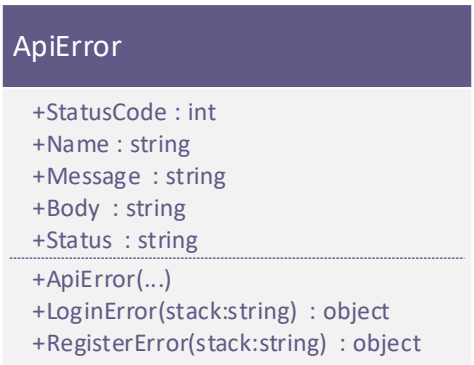


Figure 17 : Classe « ApiError »

Les méthodes statiques « LoginError » et « RegisterError » sont utilisées pour retourner facilement une instance de « ApiError » avec des paramètres prédéfinis.

4.3.4 Diagramme de séquence

Ce diagramme de séquence décrit le fonctionnement du back-end.

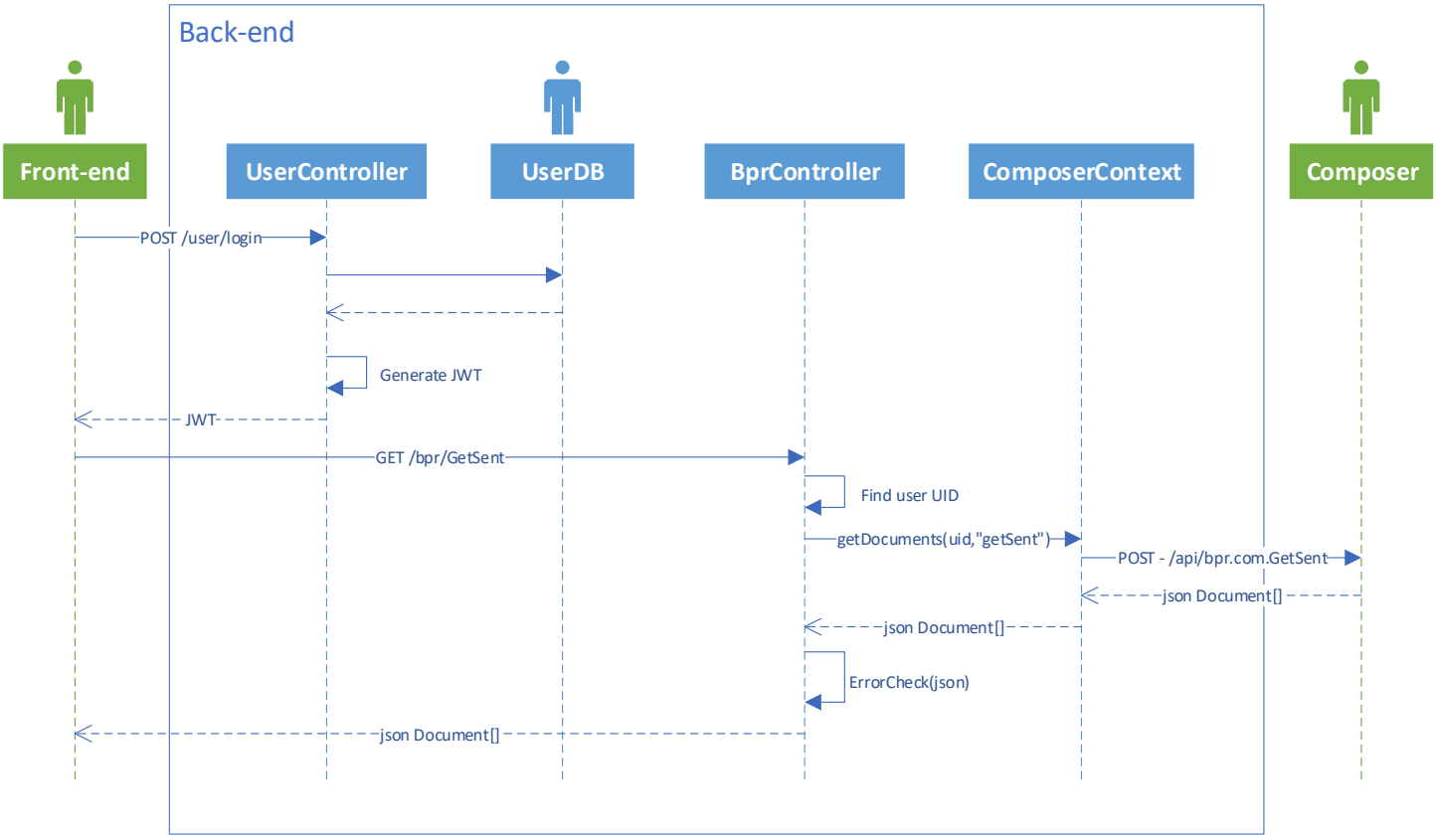


Figure 18 : Diagramme de séquence du fonctionnement du back-end

4.4 Front-End

Notre front-end est une application Web réalisée à l'aide d'Angular dans sa version 7.1.3.

4.4.1 Interface utilisateur

Pour l'interface utilisateur, nous avons décidé d'utiliser le framework CSS Bootstrap développé par Twitter. Celui-ci permet d'obtenir un résultat satisfaisant en utilisant le style par défaut. De plus, il est « responsive », ce qui signifie que l'affichage va s'adapter selon les écrans (ordinateur, tablette, téléphone portable). Il est également facilement personnalisable, ce qui permet d'adapter le design à ses besoins.

Connexion

La page de connexion contient uniquement le formulaire de connexion. Nous n'avons pas pris en charge la création de compte ainsi que l'oubli de mot de passe étant donné que ces parties seront probablement revues lors d'un passage en production (utilisateur d'un serveur LDAP par exemple).

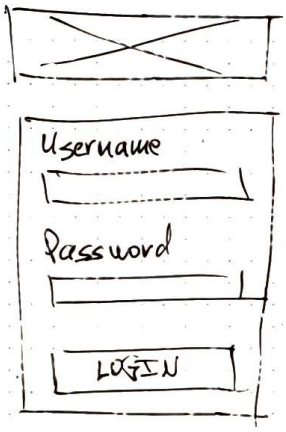


Figure 19 Mockup « Connexion »

Figure 20 Template « Connexion »

Figure 21 Message d'erreur page de connexion

Le bouton de connexion n'est pas cliquable tant que les données inscrites dans le formulaire ne sont pas valides (e-mail au bon format et mot de passe complété). Les messages d'erreurs s'affichent au-dessus du formulaire.

Dashboard

Le dashboard va rassembler la totalité des fonctionnalités en son sein, de manière à simplifier au maximum l'expérience utilisateur.

La partie de gauche va permette l'envoi rapide d'un document, tandis que la partie de droite va permettre un aperçu des documents reçus et envoyés. Un code couleur permet de connaître directement le statut d'un document (bleu : nouveau document reçu, jaune : document envoyé reçu par son destinataire, vert : document envoyé lu par son destinataire).

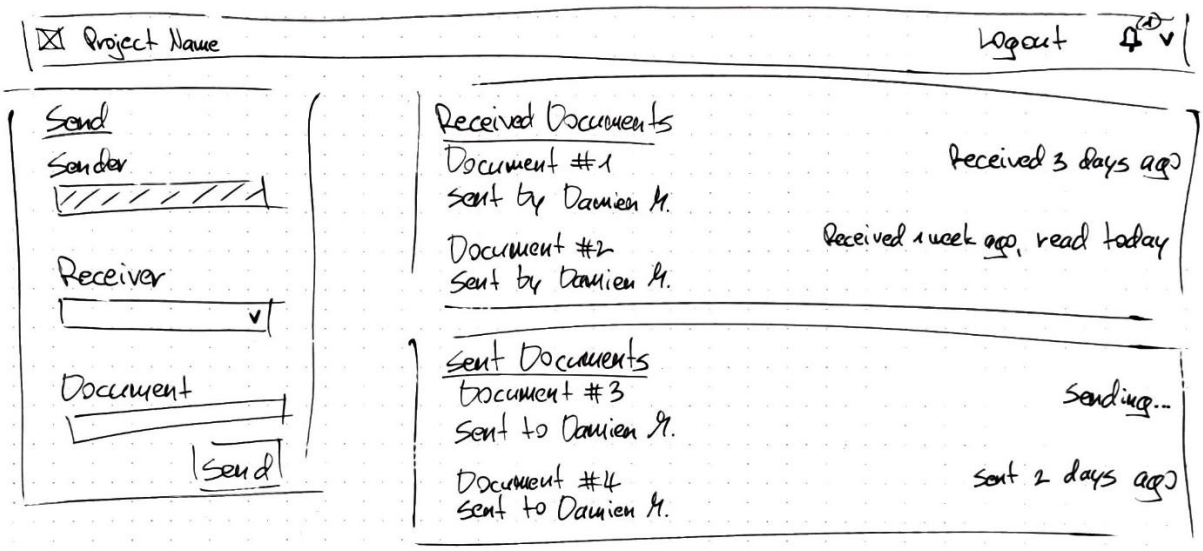


Figure 22 Mockup « Dashboard »

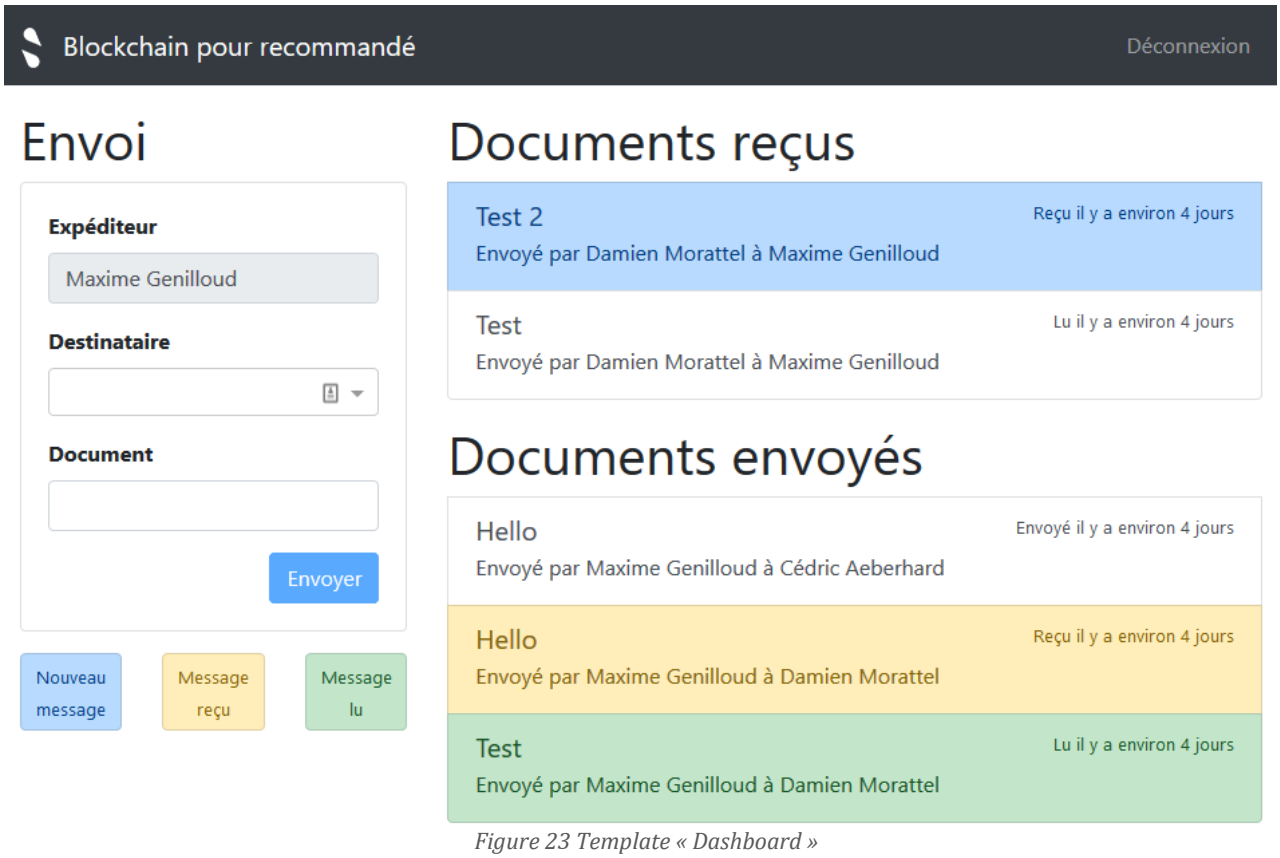


Figure 23 Template « Dashboard »

L'affichage du temps de manière relative est réalisé à l'aide du module ngx-timeago (documentation : [8]).

Les messages d'erreurs (en rouge) et de succès (en vert) s'affichent en bas à droite de l'écran, par-dessus le reste de l'interface. Ils s'empilent les uns sur les autres. Ils disparaissent après 10 secondes et peuvent être fermés manuellement pas l'utilisateur.

Ces messages utilisent les alertes (documentation : [9]) de Bootstrap, rendues disponibles nativement dans Angular (sans avoir besoin d'inclure jQuery, la libraire Javascript utilisée par Bootstrap) grâce au module ng-bootstrap (documentation : [10]).

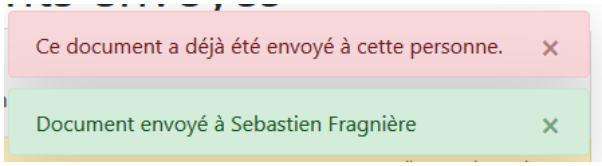


Figure 24 Messages d'erreur et de succès dashboard

Le formulaire d'envoi permet de sélectionner le ou les destinataire(s) parmi une liste d'utilisateurs. Une recherche est effectuée dans la liste lors de la saisie dans le champ. Plusieurs destinataires peuvent être sélectionnés (à savoir qu'une transaction blockchain par utilisateur sera réalisée). Ce champ est réalisé à l'aide du module ng-select (documentation : [11]).

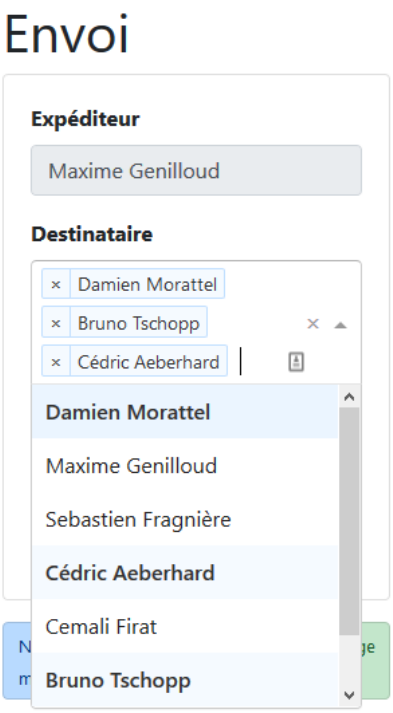


Figure 25 Liste déroulante des destinataires

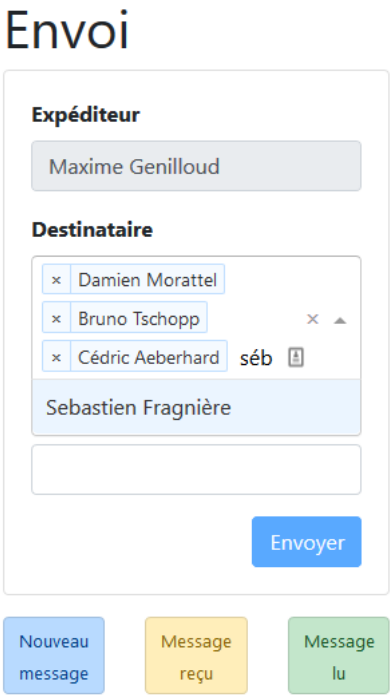


Figure 26 Recherche dans la liste

Document

Lors du clic sur un document dans la liste, les détails de celui-ci seront affichés dans une fenêtre modale (documentation : [12]). L'utilité de cette fenêtre est principalement de déclencher l'ajout du statut « READ » au document dans la blockchain. On peut également y retrouver l'historique des statuts.

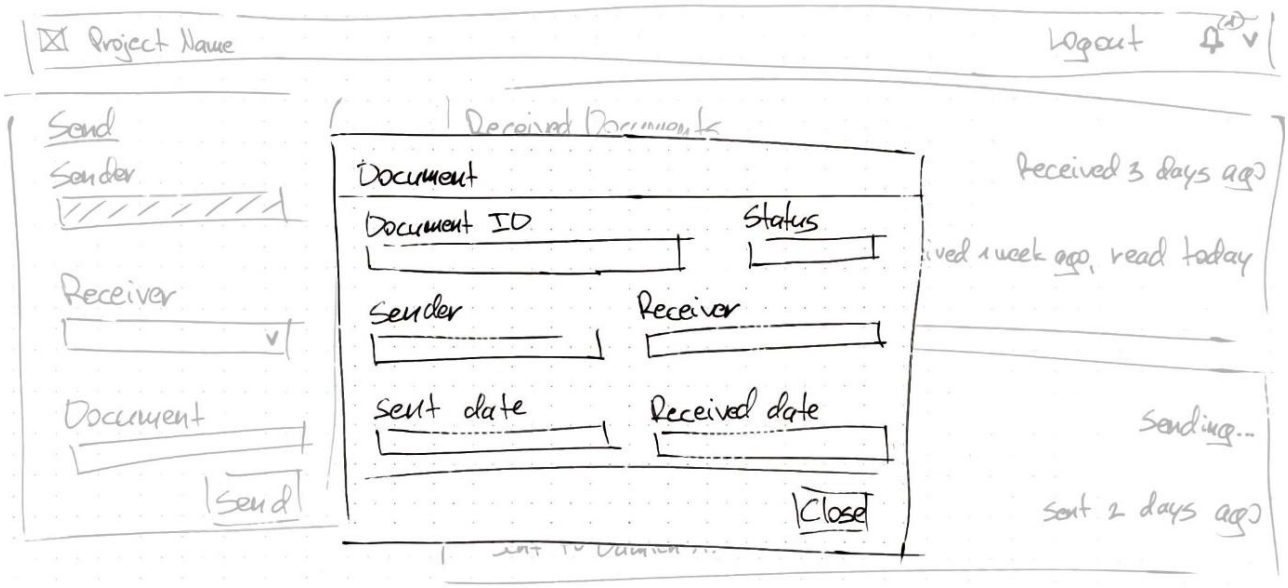


Figure 27 Mockup « Document »

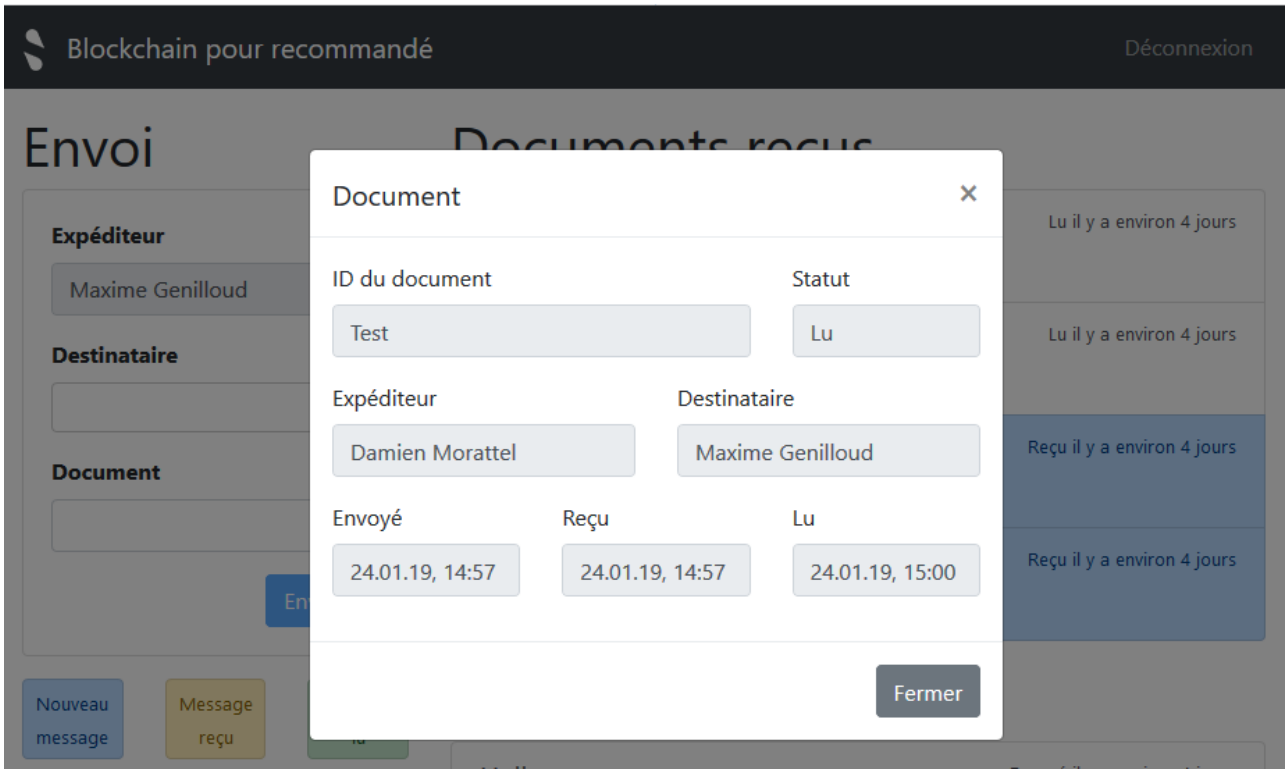


Figure 28 Template « Document »

4.4.2 Composants

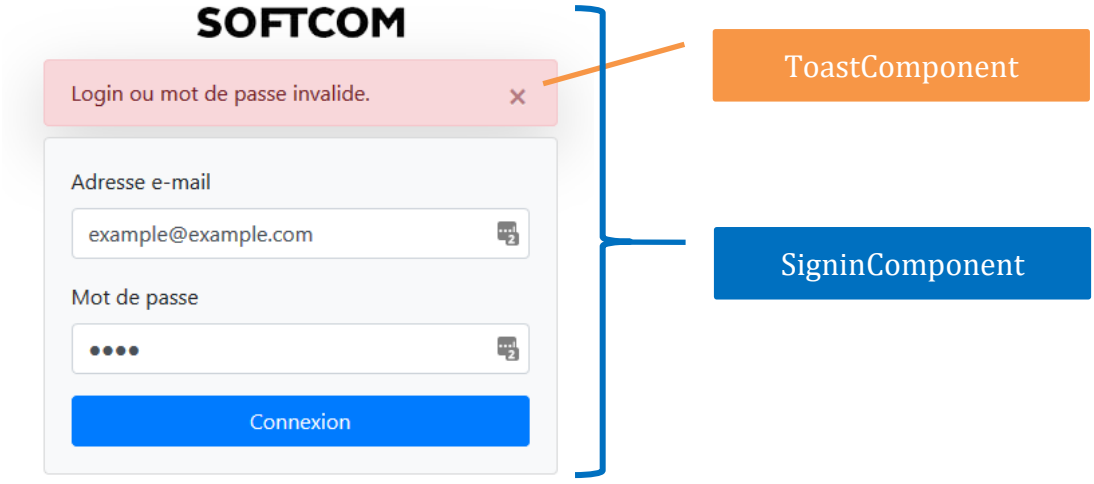
Une application Angular est divisée en composants. Chaque composant possède une vue (fichier .html), une feuille de style (fichier .scss) ainsi qu'une logique (fichier .ts). Un fichier contenant des instructions de test unitaire peut également être présent (fichier .spec.ts).

Cette séparation en composants permet une meilleure flexibilité. Il est possible de réutiliser un composant dans plusieurs vues, d'afficher un composant selon certaines conditions ou encore de réaliser une boucle sur un composant.

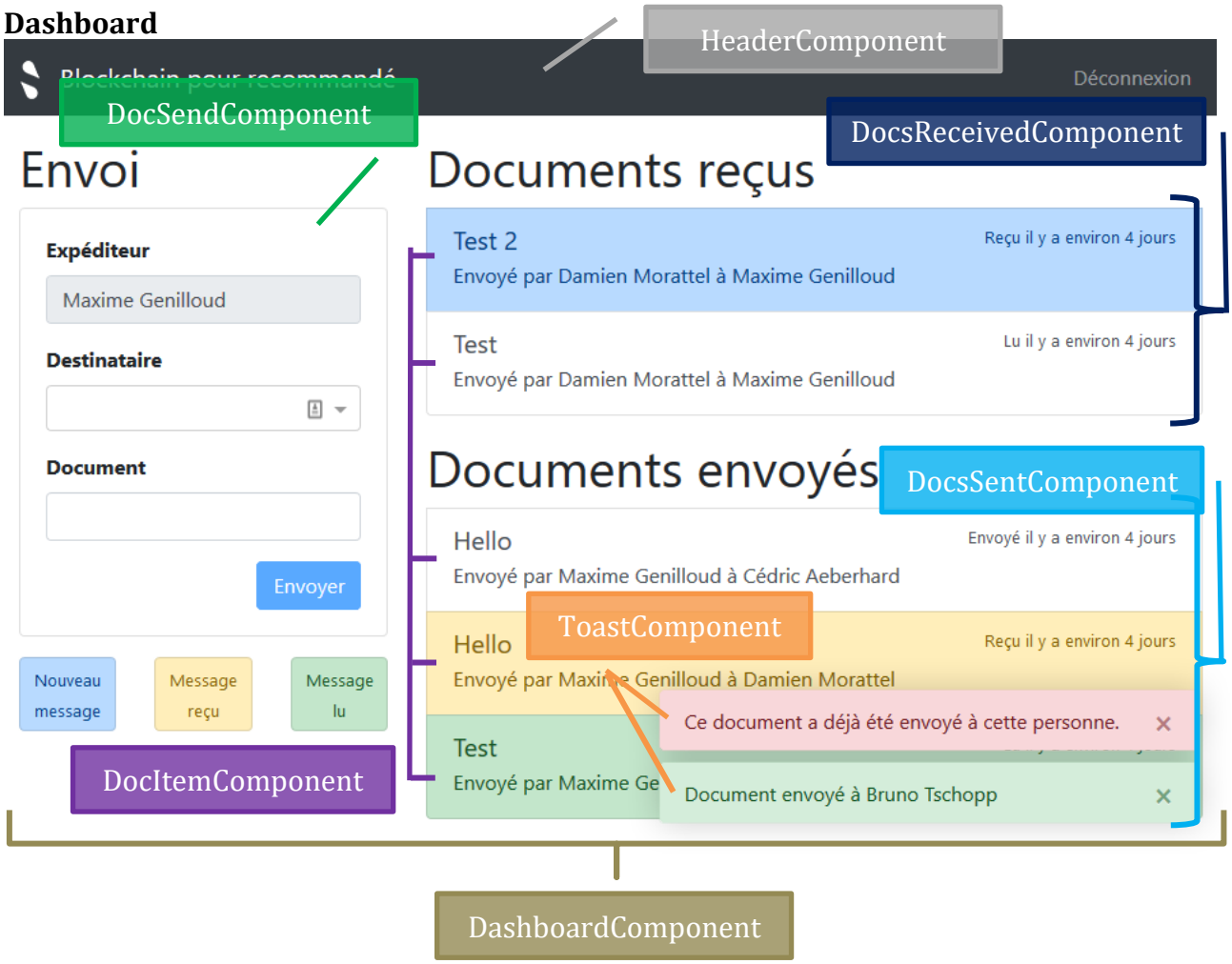
Ces composants se trouvent dans `src/app` et sont rangés dans des sous-dossiers. Voici la structure des composants de notre application (les composants sont en couleur) :

```
app
├──auth
│   └──signin
├──dashboard
├──document
│   ├──doc-item
│   ├──doc-send
│   ├──docs-received
│   └──docs-sent
├──header
└──toast
```

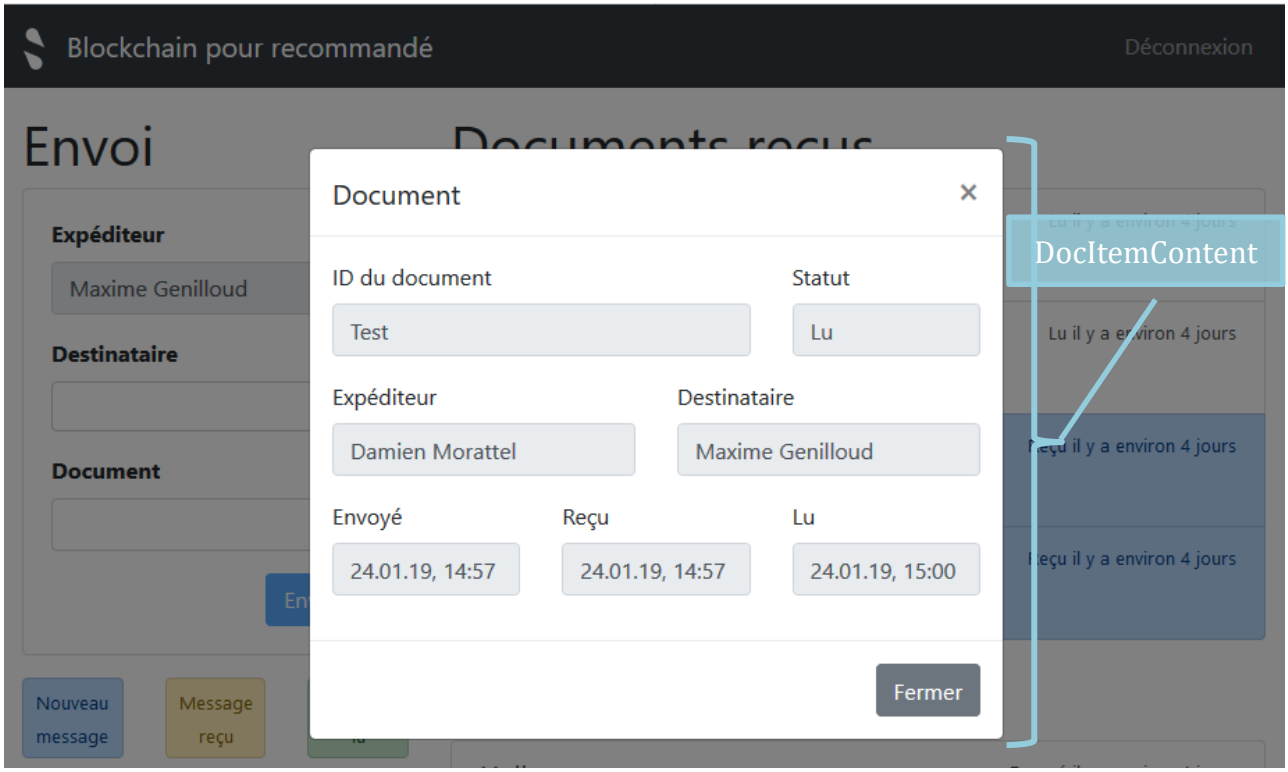
Connexion



Dashboard



Document



4.4.3 Modèles

User

Le modèle « User » contient les différents attributs d'un utilisateur. Le token est un attribut facultatif qui est utilisé uniquement pour l'utilisateur authentifié.

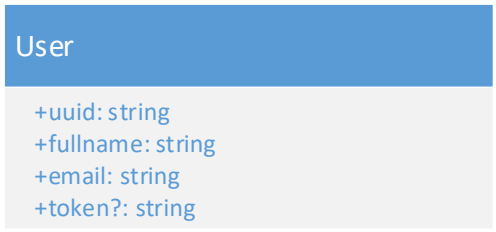


Figure 29 Modèle « User »

Document

Nous avions réalisé un modèle pour les documents, mais celui-ci était outrepasser par les réponses fournies par l'API de Composer. Nous l'avons donc supprimé.

Status

Le modèle « Status » contient une énumération des différents statuts possibles d'un document ainsi qu'un texte associé.

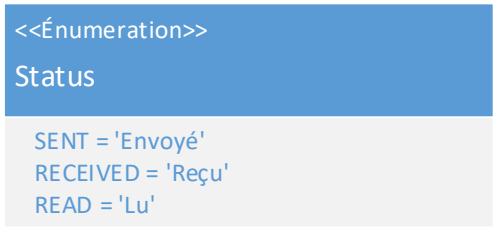


Figure 30 Modèle « Status »

Toast

Le modèle « Toast » est utilisé pour les messages d'erreur, succès et autre. Il contient un attribut « message » pour le texte, un attribut « type » pour son type et un attribut « delay » facultatif pour définir après combien de millisecondes il disparaît. Si ce dernier n'est pas précisé, le toast va rester affiché tant que l'utilisateur ne le ferme pas manuellement.

Une énumération « ToastType » est également définie pour les différents types de toast. Il est associé aux classes Bootstrap correspondantes afin d'y ajouter le style respectif.

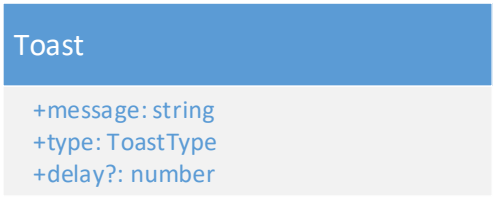


Figure 31 Modèle « Toast »

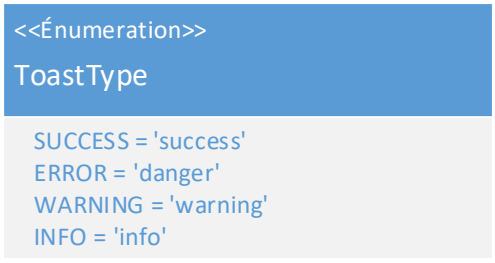


Figure 32 Modèle « ToastType »

4.4.4 Services

Un service va permettre de centraliser et partager des données ainsi que des fonctions entre différents composants. Cela permet d'éviter d'avoir du code à double et de se retrouver avec les mêmes données stockées dans plusieurs parties de l'application.

UserService
Le « UserService » va permettre de récupérer la liste des utilisateurs depuis le back-end. Il va stocker les utilisateurs dans un tableau auquel on peut souscrire à l'aide d'un « Subject ». Celui-ci va principalement être utilisé pour sélectionner un ou des destinataires depuis le formulaire d'envoi.

La méthode « getUserFullname » va également permettre de retrouver le nom complet d'un utilisateur à partir de son identifiant.

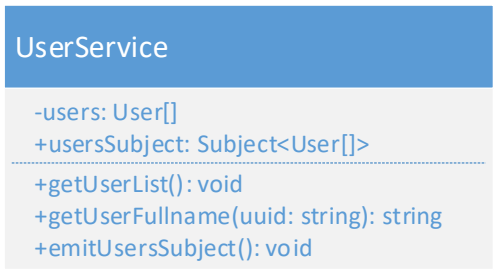


Figure 33 Service « UserService »

AuthService
Le « AuthService » va se charger de la connexion et la déconnexion d'un utilisateur.

A la connexion, l'objet « User » de l'utilisateur sera stocké dans son local storage afin qu'il puisse à nouveau accéder à l'application sans devoir se connecter à nouveau. Ce n'est pas la meilleure manière de procéder, mais c'est suffisant pour notre proof-of-concept. Comme mentionné précédemment, nous ne nous sommes pas concentrés sur l'authentification des utilisateur étant donné que cela sera géré complètement différemment en production.

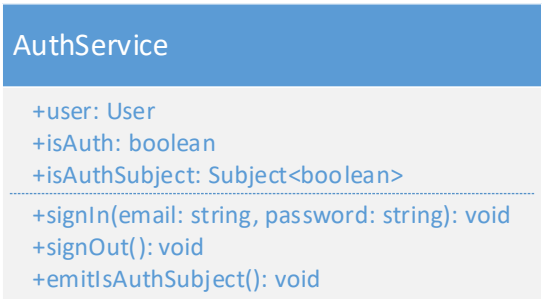


Figure 34 Service « AuthService »

AuthGuardService

Le « AuthGuardService » est là pour empêcher l'accès aux parties protégées de l'application si aucun utilisateur n'est connecté. Dans notre cas, il n'est utilisé que pour le dashboard.

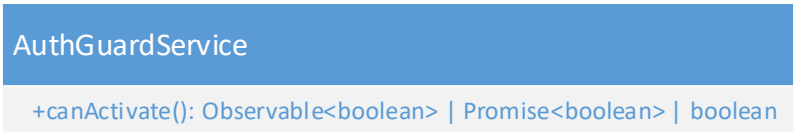


Figure 35 Service « AuthGuardService »

DocumentService

Le « DocumentService » va gérer tout ce qui concerne les documents : lister les documents reçus, lister les documents envoyés, envoyer des documents et définir un document comme étant lu.

Figure 36 Service « DocumentService »

Une méthode « showError(error) » a été créée afin de gérer les erreurs. En effet, comme vu au chapitre 4.2.5, les erreurs communiquées par l'API Composer ont un format spécial.

Cette méthode va essayer d'afficher `error.error.message` de l'objet `error`. Si elle n'y arrive pas, elle va se contenter d'afficher `error.message` qui est toujours existant et contient le message d'erreur HTTP.

Elle va également afficher une erreur spécifique lorsqu'un document a déjà été envoyé à quelqu'un. En effet, le message renvoyé par l'API Composer n'est pas très parlant. Il serait certainement plus adapté de gérer cela dans le back-end, mais nous avons manqué de temps.

ToastService

Le ToastService va s'occuper de gérer les messages à afficher à l'écran.

Figure 37 ToastService

4.5 Problèmes rencontrés

Ce chapitre répertorie les différents problèmes rencontrés au fur et à mesure du développement de l'application.

4.5.1 CORS

Nous avons rencontré des difficultés concernant la communication entre le front-end et le back-end. Les navigateurs actuels vérifient les entêtes CORS (Cross-Origin Resource Sharing [13]). Cette fonctionnalité permet de limiter à quelle(s) ressource(s) externe(s) une application Web peut-accéder.

Par défaut, les requêtes externes sont bloquées. Ce paramètre doit être défini du côté de l'API appelée. Il a donc fallu, dans le code du back-end, autoriser ces communications en ajoutant les bonnes entêtes CORS dans les réponses HTTP.

4.5.2 MVCC_READ_CONFLICT

Lors du développement du front-end, nous nous sommes heurtés à un problème du côté blockchain. Une erreur de type « MVCC_READ_CONFLICT » survenait lorsqu'un document est envoyé à un utilisateur actuellement connecté. Cette erreur s'affichait du côté du destinataire.

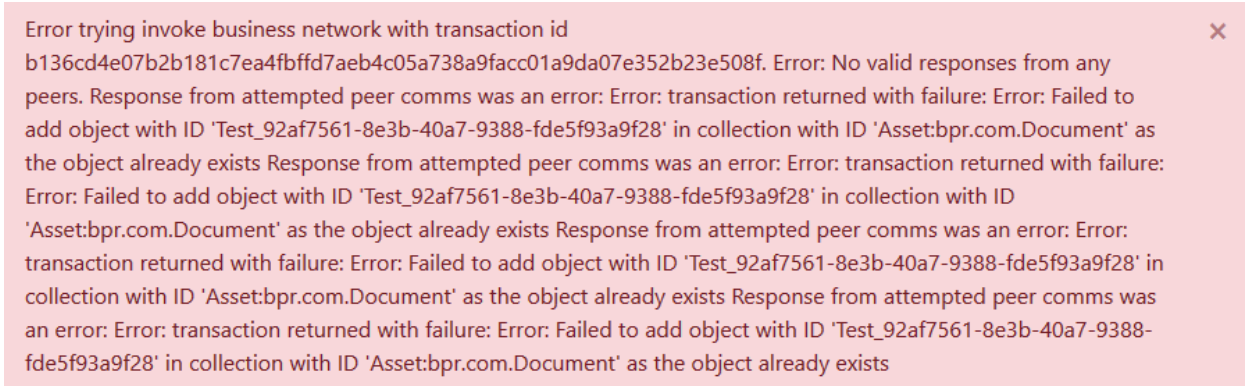


Figure 38 Message d'erreur « MVCC_READ_CONFLICT »

Celui-ci va récupérer ses documents reçus à l'aide de la méthode « GetReceived » du smart contract. Ce dernier se charge d'ajouter le statut « RECEIVED » aux documents qui ne l'ont pas encore. Cela créé un conflit du fait que les données sont encore en cours d'écriture sur les différentes peers du réseau blockchain

Une transaction est validée suite à une simulation sur les peers, mais elle n'est pas directement réalisée (commit). Ceci est un comportement connu (source : [13]) dont nous n'avions pas connaissance au début du projet.

Il existe plusieurs solutions (source : [14]) pour éviter cette erreur, mais la plupart nécessite de grosses modifications. C'est pourquoi nous avons décidé de simplement masquer cette erreur du côté front-end. Le statut « RECEIVED » sera ajouté lors de la prochaine invocation de la méthode « GetReceived » (au prochain polling de données).

4.6 Futur et améliorations

Ce chapitre présente les différentes améliorations possibles. Ce sont des points que nous n'avons malheureusement pas eu le temps de mettre en place dans le cadre du projet ou qui ne faisaient pas sens dans la réalisation d'un proof-of-concept.

4.6.1 Contrôles niveau Blockchain

Nous avons mis en place certains contrôles, comme par exemple, la vérification qu'un document n'ait pas déjà été envoyé à un utilisateur grâce à un identifiant unique.

Cependant, d'autres sécurités peuvent être mise en place, comme la vérification du format de l'identifiant du document à l'aide d'une expression régulière.

Il pourrait également être intéressant d'activer l'authentification à l'API fournie afin d'ajouter une couche de sécurité supplémentaire.

4.6.2 Front-end mis à jour en temps réel

Notre application réalise du polling de données vers le back-end afin de mettre à jour les informations de l'interface utilisateur telles que la liste des document reçus et envoyés.

Une grande amélioration pour l'expérience utilisateur serait d'utiliser un système de communication en temps réel entre le front-end et le back-end. Ceci est possible grâce à des sockets ou une API plus développée comme SignalR par exemple. Cela permettrait également d'éviter la surcharge de requêtes simultanées vers le back-end lorsqu'une masse d'utilisateurs utilisent l'application en même temps.

Cette amélioration permettrait de profiter du système d'événement d'Hyperledger Composer afin d'informer directement le front-end d'un nouvel envoi.

De plus, cela pourrait éventuellement régler l'erreur « MVCC_READ_CONFLICT » rencontrée lors de la récupération des documents (voir chapitre 4.5.2).

4.6.3 Authentification des utilisateurs

Nous utilisons les JWT (JSON Web Token) afin de contrôler l'accès à notre back-end REST. Cette solution n'est pas idéale pour l'authentification d'utilisateurs. En effet, un JWT n'est pas fait pour être invalidé manuellement du côté du serveur par exemple pour réaliser la déconnection d'un utilisateur.

La mise en place d'un front-end real time avec SignalR permettrait de résoudre ces problèmes.

4.6.4 Trier les documents par date d'envoi

Actuellement, les documents sont triés à l'aide de leur identifiant, formé du document ID ainsi que de l'ID du destinataire. Ainsi, les documents s'affichent dans un ordre complétement aléatoire.

Il pourrait être intéressant de trier les documents par date d'envoi par exemple. Le faire directement au niveau du smart contract serait idéal. En effet, cela permettrait également

d'optimiser la procédure de réception des documents qui va se charger d'ajouter le statut « RECEIVED ». Dès qu'un document avec ce statut passe dans la boucle qui parcourt tous les documents reçus par un utilisateur, on pourrait sortir de celle-ci (break).

Cette modification nécessiterait l'ajout d'un timestamp au niveau de l'asset « Document » étant donné qu'il n'est pas possible d'aller lire les entrées du tableau « Events » dans une requête. De plus, ce champ devra être défini comme sort index manuellement dans CouchDB (source : [15]).

4.6.5 Traitement des erreurs

Actuellement les erreurs générées par Composer (par exemple pour un document envoyé en double), sont directement retransmises, sans traitement.

Etant donné que les messages d'erreur fournis par Composer ne sont pas suffisamment vulgarisé pour un utilisateur lambda, le front-end s'occupe de les interpréter et de retranscrire un message intelligible pour l'utilisateur.

L'amélioration serait de traduire ces messages d'erreurs au niveau du back-end. Ainsi, si d'autres services viennent à utiliser cette même API, les messages d'erreur seraient unifiés.

4.6.6 Développement natif sur Hyperledger Fabric

Composer est un outil parfait pour la création d'un proof-of-concept. En effet, il permet de simplifier au maximum le développement de smart contracts en utilisant le langage Javascript. De plus, il génère directement une API REST pour interagir avec les dits smart contracts. Il permet également la génération d'une application Web Angular (fonctionnalité que nous n'avons pas utilisée durant le projet).

Malheureusement, le développement de Composer a été mis en pause le 30 août 2018 (source : [16]). Cela signifie qu'aucune nouvelle fonctionnalité ne va être développée pour cet outil. Cependant, la correction de bugs et la compatibilité avec les futures versions de Fabric vont être maintenus.

Pour une application en production, il est conseillé de développer les smart contracts directement en chaincode (développement natif Hyperledger Fabric). Ceci peut être réalisé à l'aide des langages GO ou Node.JS.

5 Rendu

Nous avons réalisé une documentation pour la mise en place de l'infrastructure blockchain ainsi que du back et front-end. Celle-ci est disponible en annexe.

Nous fournissons également une machine virtuelle Ubuntu sur laquelle l'infrastructure est déjà installée. Une documentation concernant cette machine est également disponible en annexe.

5.1 Structure des dossiers

Le projet est rendu dans un ZIP contenant les répertoires suivants :

Rapport	Rapport Annexes
Installation	Sources compilées Scripts d'initialisation
Machine virtuelle	Machine virtuelle Ubuntu sur laquelle l'infrastructure est installée
Procès-verbaux	Procès-verbaux des différents séances
Sources	Sources du back-end, front-end et Composer

6 Conclusion

Ce chapitre apporte des réflexions sur la totalité du projet.

6.1 Atteinte des objectifs

Les objectifs de ce projet de semestre étaient les suivants :

1. Réalisation d'un rapport d'analyse des différentes technologies blockchain potentiellement adaptées au projet.
2. Spécification du use case relatif à la validation de la réception d'un document.
3. Elaboration d'un prototype d'interface web interagissant avec la blockchain choisie et répondant au use case.

6.1.1 Rapport d'analyse

Notre analyse traite dans un premier temps de la blockchain dans sa globalité, avant d'approfondir la solution vers laquelle nous nous sommes dirigés, à savoir Hyperledger Fabric en combinaison avec Composer.

Nous avons expliqué les différents termes et concepts associés à la technologie blockchain, tels que les dApps et les smart contracts. Nous avons également réalisé un tableau comparatif des technologies nous paraissant adaptées au projet. Finalement, nous avons sélectionné la technologie sur laquelle nous avons basé notre prototype.

La blockchain choisie répondant aux différents critères fournis par le mandant, notamment au niveau de la protection des données, nous pouvons considérer cet objectif comme atteint.

6.1.2 Use case

La spécification du projet s'est réalisée en plusieurs étapes. Nous avons commencé par schématiser le fonctionnement global du prototype. Les limites du prototype ont également été clairement fixées dès le début.

Nous avons ensuite détaillé plus précisément les différents cas d'utilisation en réalisant un diagramme de use case ainsi que les fiches descriptives qui en découlent. Un diagramme de composant a également été réalisé afin de représenter les interactions entre les différents éléments qui forment le projet. Il nous a permis de partir sur de bonnes bases.

Nous avons finalement défini les technologies utilisées pour la suite du projet, que ce soit la blockchain, le back-end et le front-end.

Les différents cas d'utilisation ont été validé par nos responsables de projet ainsi que par le mandant. De plus, les technologies ayant été sélectionnées en accord avec le mandant, nous pouvons considérer cet objectif comme atteint.

6.1.3 Prototype

Afin de mettre en place l'infrastructure blockchain, nous avons collaboré avec Cyril Vallélian. Il nous a fourni une machine virtuelle sur le réseau de l'école, avec des scripts facilitant le déploiement d'Hyperledger Fabric et Composer.

Sur cette base, nous avons commencé par réaliser différents tests, avant de développer un smart contract répondant au cahier des charges.

Une fois le smart contract au point, nous avons développé en parallèle le back-end ainsi que le front-end. Ceux-ci ont été réalisés le plus simplement possible, afin de répondre au mieux aux demandes du mandat et de permettre de faire évoluer le projet par la suite.

Nous avons ensuite réalisé un test utilisateur avec le mandant, dans le but d'avoir ses retours sur notre projet. Il nous a confié être totalement satisfait par le prototype développé. D'après son feedback, nous pouvons donc considérer cet objectif comme atteint.

6.2 Conclusions personnelles

6.2.1 Maxime Genilloud

Ce projet fut passionnant ! J'ai pu découvrir le côté technique de certaines technologies qui m'ont toujours intéressé, je pense notamment à la blockchain ainsi qu'aux frameworks Javascript. J'ai également apprécié le fait d'avoir, avec mon collègue, réaliser un projet dans son ensemble de manière autonome.

De nature perfectionniste, il y a évidemment certains aspects du projet que j'aurais souhaité approfondir ou réaliser différemment. En effet, nous aurions aimé aller plus loin et réaliser certaines des améliorations proposées dans ce rapport, mais le temps étant limité, cela ne fut malheureusement pas possible.

Je conserve un très bon souvenir de cette expérience enrichissante. J'ai d'ores et déjà hâte de réaliser le prochain projet de semestre.

6.2.2 Damien Morattel

Ce projet a été très instructif pour moi, premièrement car cela faisait un certain temps que je m'intéressais à la technologie blockchain et aux smart contracts. J'ai pu approfondir mes connaissances dans ces domaines. Ayant travaillé plus sur le côté back-end du proof-of-concept, j'ai également pu acquérir des connaissances sur la programmation dot NET Core et ses fonctionnalités.

Il était parfois difficile de s'imposer des limites sur le scope du prototype et de la documentation, mais cela fut nécessaire afin de terminer le projet dans le temps alloué. J'aurai par exemple beaucoup aimé voir en détail la programmation native Hyperledger sans utiliser Composer. Ou encore mettre en place une communication « live » entre le front-end et le back-end afin de gérer les événements.

7 Déclaration d'honneur

Nous, soussignés, Maxime Genilloud et Damien Morattel, déclarons sur l'honneur que le travail rendu est le fruit d'un travail personnel. Nous certifions ne pas avoir eu recours au plagiat ou à toutes autres formes de fraudes. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.

Damien Morattel	Maxime Genilloud

8 Glossaire

API	Façade par laquelle une application offre ses services.
API REST	API fonctionnant avec des requêtes HTTP (POST, GET, ...).
Back-end	Partie de l'application traitant les données pour les retourner au front-end.
Block	Ensemble de transactions.
Blockchain	Livre de compte décentralisé.
BNA (Business Network Archive)	Archive contenant les fichiers du BND.
BND (Business Network Definition)	Ensemble de fichiers modélisants et décrivant la logique d'un smart contract généré avec Composer.
Chaincode	Smart contract. Nom utilisé par Hyperledger.
Consensus	Procédure grâce à laquelle un réseau de nœuds va pouvoir garantir la validité des transactions.
Crowdfunding	Financement participatif.
dApp	Application décentralisée.
Front-end	Partie de l'application avec laquelle l'utilisateur final interagit.
JSON	Méthode de formatage de données.
LDAP	Système de gestion d'utilisateurs et de groupes sous forme d'un annuaire hiérarchique.
Mineur	Nœud qui s'occupe de vérifier la validité des transactions blockchain afin de les ajouter au dernier block.
Noeud	Ordinateur/serveur faisant partie d'un réseau blockchain.
PoS (Proof Of Stake)	Méthode de consensus par preuve d'enjeu.
PoW (Proof Of Work)	Méthode de consensus par calcul cryptographiques.
Smart contract	Script stocké et exécuté sur une blockchain par transaction.
Token	Avoir virtuel sur la blockchain.
Transactions	Une transaction est créée lors de l'appel d'un smart contract. Elle contient les données ajoutées ou modifiées de la blockchain.
World state	Base de données CouchDB contenant le dernier état des éléments stockés sur la blockchain.
PoC (Proof-of-Concept)	Application prototype permettant de démontrer la faisabilité et le fonctionnement d'un concept.

9 Références

9.1 Figures

Figure 1 : Différence entre une application Web et une dApp 9
Figure 2 : Schéma du fonctionnement de Composer. Source : [1] 13
Figure 3 : Schéma du fonctionnement global du proof-of-concept 16
Figure 4 : Diagramme des états d'un envoi 17
Figure 5 : Diagramme de cas d'utilisations 18
Figure 6 : Diagramme des composants 18
Figure 7 Diagramme MVVM. Source : [4] 20
Figure 8 : Structure des fichiers du BND 21
Figure 9 : Structure des données définie dans le fichier bpr.com.cto 22
Figure 10 : Transactions définie dans le fichier bpr.com.cto 22
Figure 11 : Schéma des actions REST de Composer 26
Figure 12 : Diagramme de classe du back-end 28
Figure 13 : Classe du contrôleur "UserController" 29
Figure 14 : Schéma des actions REST de UserController 30
Figure 15 : Classes utilisées pour les interactions avec Composer 31
Figure 16 : Schéma des actions REST de BprController 32
Figure 17 : Classe « ApiError » 32
Figure 18 : Diagramme de séquence du fonctionnement du back-end 33
Figure 19 Mockup « Connexion » 34
Figure 20 Template « Connexion » 34
Figure 21 Message d'erreur page de connexion 34
Figure 22 Mockup « Dashboard » 35
Figure 23 Template « Dashboard » 35
Figure 24 Messages d'erreur et de succès dashboard 36
Figure 25 Liste déroulante des destinataires 36
Figure 26 Recherche dans la liste 36
Figure 27 Mockup « Document » 37
Figure 28 Template « Document » 37
Figure 29 Modèle « User » 40
Figure 30 Modèle « Status » 40
Figure 31 Modèle « Toast » 40
Figure 32 Modèle « ToastType » 40
Figure 33 Service « UserService » 41
Figure 34 Service « AuthService » 41
Figure 35 Service « AuthGuardService » 42
Figure 36 Service « DocumentService » 42
Figure 38 ToastService 43
Figure 32 Message d'erreur « MVCC_READ_CONFLICT » 44

9.2 Tableaux

Tableau 1 : Eléments constituants une blockchain 5
Tableau 2 : Comparatifs des différentes blockchains 8
Tableau 3 : Eléments de modélisation de l'outil Composer 13
Tableau 4 : Etats possible d'un document 17
Tableau 5 : Données contenu dans une transaction 19
Tableau 6 : Fichiers du BND 21
Tableau 7 : Propriétés utilisées pour décrire un utilisateur 29

9.3 Bibliographie

[1] «Documentation Hyperledger Composer,» [En ligne]. Available: https://hyperledger.github.io/composer/v0.16/introduction/introduction.html. [Accès le 26 01 2019].

[2] H. Composer, «Using Playground,» [En ligne]. Available: https://hyperledger.github.io/composer/v0.19/playground/playground-index. [Accès le 29 01 2019].

[3] W. Montes, «Arrêt du developpement de Composer,» [En ligne]. Available: https://hackernoon.com/hyperledger-composer-has-been-put-on-pause-what-are-your-options-now-dfc3a97a6308. [Accès le 26 01 2019].

[4] E. Bataille, «Un modèle d'architecture AngularJS,» [En ligne]. Available: https://www.technologies-ebusiness.com/enjeux-et-tendances/modele-darchitecture-angularjs. [Accès le 28 01 2019].

[5] «Welcome to Hyperledger Composer | Hyperledger Composer,» [En ligne]. Available: https://hyperledger.github.io/composer/latest/introduction/introduction. [Accès le 28 11 2018].

[6] «Emitting Events | Hyperledger Composer,» [En ligne]. Available: https://hyperledger.github.io/composer/v0.16/business-network/publishing-events. [Accès le 29 01 2019].

[7] «Hyperledger Composer REST Server | Hyperledger Composer,» [En ligne]. Available: https://hyperledger.github.io/composer/v0.19/reference/rest-server. [Accès le 29 01 2019].

[8] «ihym/ngx-timeago: Live updating timestamps in Angular 6+,» [En ligne]. Available: https://github.com/ihym/ngx-timeago. [Accès le 12 12 2018].

[9] «Alerts - Bootstrap,» [En ligne]. Available: https://getbootstrap.com/docs/4.0/components/alerts/. [Accès le 12 12 2018].

[10] «Angular powered Bootstrap,» [En ligne]. Available: https://ng-bootstrap.github.io. [Accès le 12 12 2018].

[11] «ng-select/ng-select: Native angular 6+ select component,» [En ligne]. Available: https://github.com/ng-select/ng-select. [Accès le 12 12 2018].

[12] «Modal - Bootstrap,» [En ligne]. Available: https://getbootstrap.com/docs/4.0/components/modal/. [Accès le 12 12 2018].

[13] M. W. docs, «Cross-Origin Resource Sharing (CORS),» [En ligne]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Accès le 29 01 2019].

[14] «High-Throughput Network,» [En ligne]. Available: https://github.com/hyperledger/fabric-samples/tree/release/high-throughput/. [Accès le 23 01 2019].

[15] «Hyperledger Fabric and how it isn't concurrent out of the box.,» [En ligne]. Available: https://medium.com/wearetheledger/hyperledger-fabric-concurrency-really-eccd901e4040. [Accès le 23 01 2019].

[16] «Query Order expression is failing - Issue #1640 - hyperledger/composer,» [En ligne]. Available: https://github.com/hyperledger/composer/issues/1640. [Accès le 09 01 2019].

[17] «Composer TSC update,» [En ligne]. Available: https://lists.hyperledger.org/g/composer/message/125. [Accès le 05 12 2018].

10 Annexes

1) Cahier des charges
2) Planification effective
3) Fiches descriptives
4) Mise en place de l'infrastructure BPR
5) Machine virtuelle BPR

Annexe 1

Cahier des charges

Projet de semestre 5

La blockchain pour réaliser du recommandé

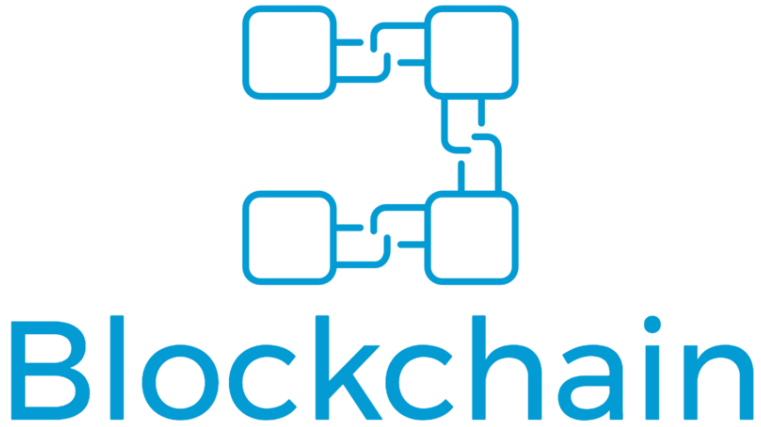
Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

La blockchain pour réaliser du recommandé

Cahier des charges

Projet de semestre 5

Filière Télécommunication – Internet et communications



Maxime Genilloud - Damien Morattel

T3F

Responsable externe : Jonathan Gilgen – Softcom

Responsables internes: Houda Chabbi
Patrick Gaillet

Version 1.0 du 11.10.2018

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Table des matières

1 Contexte 1
2 Objectifs 2
3 Activités 3
3.1 Objectif A 3
3.2 Objectif B 3
3.3 Objectif C 3
4 Planning 4

1 Contexte

L'entreprise SoftCom Technologies SA basée à Granges-Paccot est spécialisée dans le développement de logiciels sur mesure et l'intégration des services.

SoftCom souhaite proposer à ses client un système permettant l'envoi sécurisé de documents entre deux personnes. Pour cela, ils travaillent en coopération avec la startup *e-sign* de Swisscom qui a mis en place un outil permettant de signer numériquement des documents afin d'en garantir l'authenticité. Ces documents peuvent ensuite être partagés à l'aide d'une URL.

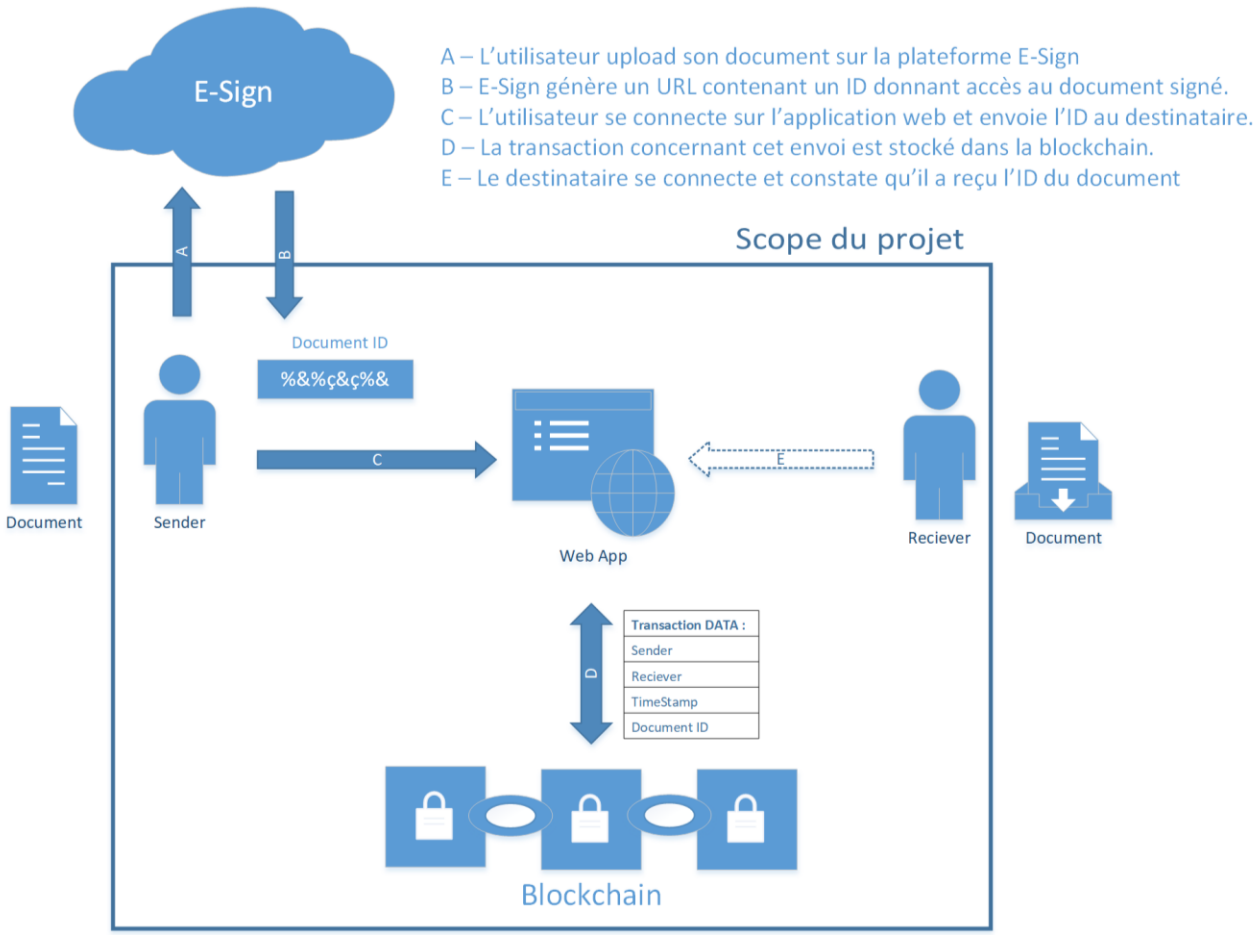
Un des aspects importants lors de cet envoi de document est de pouvoir vérifier qu'il a bien été envoyé et réceptionné par la bonne personne, tel un courrier recommandé. SoftCom nous mandate afin de trouver un cas d'utilisation de la technologie blockchain pour stocker les informations sur l'expéditeur et le destinataire de manière infalsifiable et consultable.

La technologie blockchain peut être vue comme un registre de transactions décentralisé. Les transactions sont regroupées pour former des blocks, puis un système de vérification (par exemple le minage) vérifie l'exactitude du block avant de l'intégrer à la chaine. Une fois intégré, le contenu des blocks ne peut plus être modifié.

Il existe une multitude de blockchain différentes avec chacune ses caractéristiques (transactions chiffrées ou non, vérification des blocks par proof-of-work ou proof-of-stack, accès publique ou privé, etc.).

Pour mener à bien ce projet, il faudra donc choisir la technologie et les outils blockchain les plus adaptés à la situation, afin de mettre en place un premier prototype sous la forme d'une plateforme web. Cette dernière permettra à deux utilisateurs authentifiés de se transmettre un document numérique signé stocké sur *e-sign*, en passant par la blockchain.

L'illustration ci-dessous a été réalisée afin de schématiser le projet et de faciliter sa compréhension.



2 Objectifs

A. Réalisation d'un rapport d'analyse des différentes technologies blockchain potentiellement adaptées au projet.

B. Spécification du use case relatif à la validation de la réception d'un document.

C. Elaboration d'un prototype d'interface web interagissant avec la blockchain choisie et répondant au use case.

3 Activités

3.1 Objectif A

Réalisation d'un rapport d'analyse des différentes technologies blockchain potentiellement adaptées au projet.

1) Evaluer l'utilité des dApps et des smart contracts dans le scope de notre projet.
2) Analyser différentes blockchains/frameworks adaptés au projet.
3) Réaliser un tableau comparatif contenant les données collectées.
4) Définir la technologie la plus adaptée sur laquelle nous allons baser la suite du projet.

3.2 Objectif B

Spécification du use case relatif à la validation de la réception d'un document.

1) Schématiser le use case global du projet.
2) Définir les limites de notre futur prototype.
3) Trouver une méthode d'exploitation de la blockchain pour réaliser du recommandé.
4) Définir la méthode d'authentification des utilisateurs.
5) Choisir les technologies front-end et back-end pour la réalisation du prototype.

3.3 Objectif C

Elaboration d'un prototype d'interface web interagissant avec la blockchain choisie et répondant au use case.

1) Selon la blockchain choisie, mise en place de l'infrastructure nécessaire (nœuds, serveurs, etc.)
1) Prise en main de l'API permettant d'interagir avec la blockchain.
2) Mise en place de l'infrastructure web nécessaire.
3) Réalisation de la partie back-end.
4) Design de l'interface utilisateur.
5) Réalisation de la partie front-end.
6) Test utilisateurs.
7) Adaptations et corrections de bugs.
8) Documentation

4 Planning

Planning Tâches	État	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18
ANALYSE																			
Evaluer l'utilité des dApps et des smart contracts dans le scope de notre projet																			
Analyser différentes blockchains/frameworks adaptés au projet																			
Réaliser un tableau comparatif contenant les données collectées																			
Définir la technologie la plus adaptée sur laquelle nous allons baser la suite du projet																			
SPECIFICATIONS																			
Schématiser le use case global du projet																			
Définir les limites de notre futur prototype																			
Trouver une méthode d'exploitation de la blockchain pour réaliser du recommandé																			
Définir la méthode d'authentification des utilisateurs																			
Choisir les technologies front-end et back-end pour la réalisation du prototype																			
REALISATION																			
Selon la blockchain choisie, mise en place de l'infrastructure nécessaire																			
Prise en main de l'API permettant d'interagir avec la blockchain																			
Mise en place de l'infrastructure web nécessaire																			
Réalisation de la partie back-end																			
Design de l'interface utilisateur																			
Réalisation de la partie front-end																			
Test utilisateurs																			
Adaptations et corrections de bugs																			
Documentation																			
JALONS																			
Choix de la technologie de blockchain à utiliser																			
Transaction sur la blockchain fonctionnel																			
DATES CLES																			
Début du projet																			
Rendu du cahier des charges																			
Présentation intermédiaire																			
Rendu du rapport																			
Défence orale																			

Légende:
Damien Morattel
Maxime Genilloud
Damien et Maxime
Tâche réelement réalisée -> X

Annexe 2

Planification effective

Projet de semestre 5

La blockchain pour réaliser du recommandé

Planning Tâches	État	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18
ANALYSE																			
Evaluer l'utilité des dApps et des smart contracts dans le scope de notre projet						x													
Analyser différentes blockchains/frameworks adaptés au projet								x											
Réaliser un tableau comparatif contenant les données collectées								x											
Définir la technologie la plus adaptée sur laquelle nous allons baser la suite du projet								x	x	x									
SPECIFICATIONS																			
Schématiser le use case global du projet					x		x												
Définir les limites de notre futur prototype					x														
Trouver une méthode d'exploitation de la blockchain pour réaliser du recommandé												x							
Définir la méthode d'authentification des utilisateurs									x										
Choisir les technologies front-end et back-end pour la réalisation du prototype									x										
REALISATION																			
Selon la blockchain choisie, mise en place de l'infrastructure nécessaire											x								
Prise en main de l'API permettant d'interagir avec la blockchain												x	x						
Mise en place de l'infrastructure web nécessaire														x					
Réalisation de la partie back-end													x	x	x	x			
Design de l'interface utilisateur										x	x								
Réalisation de la partie front-end													x	x	x	x			
Test utilisateurs																	x		
Adaptations et corrections de bugs															x	x		x	x
Documentation					x	x	x	x	x	x	x				x	x	x	x	x
JALONS																			
Choix de la technologie de blockchain à utiliser																			
Transaction sur la blockchain fonctionnel																			
DATES CLES																			
Début du projet																			
Rendu du cahier des charges																			
Présentation intermédiaire																			
Rendu du rapport																			
Défence orale																			

Légende:
Damien Morattel
Maxime Genilloud
Damien et Maxime
Tâche réelement réalisée -> X

Annexe 3

Fiches descriptives

Projet de semestre 5

La blockchain pour réaliser du recommandé

Authenticate

1 : Sommaire d'identification :

Résumé : Authentification d'un utilisateur
Acteurs : User

Auteur : Maxime Genilloud et Damien Morattel
Date de création : 31.01.2019 **Modifications** : -
Version : 1.0 **Responsable** : Jonathan Gilgen

2 : Description des enchaînements :

Pré conditions : -

Scénario nominal

1. L'utilisateur se rend sur une page de l'application Web
2. Le système le redirige vers la page de connexion
3. L'utilisateur entre ses informations de connexion (nom d'utilisateur et mot de passe)
4. L'utilisateur envoie le formulaire
5. Le système vérifie les informations d'après sa base de données
6. Le système créé une session pour l'utilisateur
7. Le système redirige l'utilisateur vers la page demandée

Enchaînements alternatifs

A1 : Utilisateur déjà connecté

- Démarre au point 1 du scénario nominal
  1. L'utilisateur est déjà connecté
  2. Reprend au point 6 du scénario nominal

Enchaînements d'exception

E1 : Informations de connexion invalides

- Démarre au point 5 du scénario nominal
  1. Un message d'erreur s'affiche sur la page de connexion
  2. Reprend au point 3 du scénario nominal

Post conditions :

3 : Besoins d'IHM (optionnel)

4 : Remarques complémentaires (optionnel)

Send Document ID

1 : Sommaire d'identification :

Résumé : Envoi de l'ID d'un document d'un utilisateur à un autre
Acteurs : User et Blockchain

Auteur : Maxime Genilloud et Damien Morattel
Date de création : 31.01.2019 **Modifications** : -
Version : 1.0 **Responsable** : Jonathan Gilgen

2 : Description des enchaînements :

Pré conditions : -

Scénario nominal

1. « include Authenticate »
2. L'utilisateur entre l'ID du document dans le champ dédié
3. L'utilisateur choisi un destinataire dans une liste déroulante
4. L'utilisateur envoie le formulaire
5. Le système envoie la requête à la blockchain
6. Le système affiche l'état de l'envoi en temps réel

Enchaînements alternatifs

-

Enchaînements d'exception

E1 : Les entrées ne sont pas valides

- Démarre au point 4 du scénario nominal
  1. Un message d'erreur s'affiche
  2. Reprend au point 1 du scénario nominal

Post conditions : -

3 : Besoins d'IHM (optionnel)

4 : Remarques complémentaires (optionnel)

Check received IDs

1 : Sommaire d'identification :

Résumé : Réception des documents reçus par l'utilisateur
Acteurs : User et Blockchain

Auteur : Maxime Genilloud et Damien Morattel
Date de création : 31.01.2019 **Modifications** : -
Version : 1.0 **Responsable** : Jonathan Gilgen

2 : Description des enchaînements :

Pré conditions : -

Scénario nominal

1. « include Authenticate »
2. Le système affiche la liste des documents reçus pour l'utilisateur connecté

Enchaînements alternatifs

A1 : L'utilisateur clique sur un document

- Démarre au point 2 du scénario nominal
  1. Le système affiche le document dans un nouvel onglet
  2. Reprend au point 1 du scénario nominal

Enchaînements d'exception

E1 : Aucun document n'a été reçu

- Démarre au point 2 du scénario nominal
  1. Le système affiche un message d'information
  2. Reprend au point 1 du scénario nominal

Post conditions : -

3 : Besoins d'IHM (optionnel)

4 : Remarques complémentaires (optionnel)

Logout

1 : Sommaire d'identification :

Résumé : Déconnexion d'un utilisateur
Acteurs : User

Auteur : Maxime Genilloud et Damien Morattel
Date de création : 31.01.2019 **Modifications** : -
Version : 1.0 **Responsable** : Jonathan Gilgen

2 : Description des enchaînements :

Pré conditions : -

Scénario nominal

1. L'utilisateur clique sur le lien de déconnexion
2. Le système détruit la session de l'utilisateur
3. Le système redirige l'utilisateur vers la page de connexion

Enchaînements alternatifs

-

Enchaînements d'exception

E1 : L'utilisateur n'était pas connecté

- Démarre au point 1 du scénario nominal
    1. Reprend au point 3 du scénario nominal

Post conditions : -

3 : Besoins d'IHM (optionnel)

4 : Remarques complémentaires (optionnel)

Annexe 4

Mise en place de l'infrastructure BPR

Projet de semestre 5

La blockchain pour réaliser du recommandé

Mise en place de l'infrastructure BPR

1 Installation

Effectué sur Ubuntu 16.04.5 Server amd64.

Les commandes sans le « sudo » explicitement spécifié sont à exécuter telles quelles (sans sudo).

1.1 Prérequis

https://hyperledger.github.io/composer/latest/installing/installing-prereqs#ubuntu

Installer les prérequis :

```
curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh

chmod u+x prereqs-ubuntu.sh

./prereqs-ubuntu.sh
```

Installer les modules python3 Jinja2 et Colorama :

```
sudo apt-get -y install python3-pip

pip3 install Jinja2

pip3 install colorama
```

1.2 Composer

https://hyperledger.github.io/composer/latest/installing/development-tools

Installer les outils Composer :

```
npm install -g composer-cli@0.20

npm install -g composer-rest-server@0.20
```

1.3 Hyperledger

https://hyperledger-fabric.readthedocs.io/en/release-1.4/install.html

Installer Hyperledger dans le dossier home de l'utilisateur :

```
curl -sSL http://bit.ly/2ysbOFE | bash -s 1.2.1
```

1.4 Scripts

Copier le fichier « bpr.tar.gz » dans le dossier de l'utilisateur et l'extraire avec :

```
tar -xvzf bpr.tar.gz
```

Copier le dossier « /bin » depuis « ~/fabric-samples » vers le dossier « ~/bpr/hyperledger » :

```
cp -r fabric-samples/bin/ bpr/hyperledger/
```

2 Utilisation

Cette partie explique comment utiliser l'infrastructure.

2.1 Initialisation

Lancer le script de provisioning en indiquant le nombre de pairs voulu :

```
python3 fabric_provisioning.py 1
```

Rendre les nouveaux scripts exécutables :

```
find ./ -type f -iname "*.sh" -exec chmod +x {} \;
```

Initialiser le réseau Fabric :

```
./init.sh
```

Ce script « init.sh » va arrêter tous les dockers qui tournent sur la machine ; il faudra donc relancer le back-end et le front-end avec leurs scripts respectifs.

Il supprime également toute la blockchain. Les données stockées seront donc perdues.

Le code Composer du BND (Business Network Definition) est appliqué à la blockchain lors du init.

2.2 Utilisation

2.2.1 Blockchain

Les scripts « start.sh » et « stop.sh » disponibles dans le dossier « bpr/ » permettent de lancer ou d'arrêter le réseau Fabric. Le réseau est directement lancé après l'utilisation du script « init.sh ».

L'API REST de Composer peut être lancée grâce au script « start_rest.sh » disponible dans le dossier « bpr/composer ».

2.2.2 Back-end

Utiliser le script « init.sh » disponible dans le dossier « bpr/backend » afin de créer et lancer les dockers du back-end. Le back-end sera disponible sur le port HTTPS 5001 et HTTP 5000.

```
cd backend/

./init.sh
```

2.2.3 Front-end

Utilisez le script « init.sh » disponible dans le dossier « bpr/frontend » afin de créer et lancer les dockers du front-end. Le front-end sera disponible sur le port HTTP 8888.

```
Cd frontend/

./init.sh
```

Le fichier de configuration « bpr/frontend/angular/assets/config.json » permet de changer l'adresse du serveur du back-end si celui-ci est sur une autre machine. Il permet aussi de changer d'autres paramètres telles que l'intervalle de polling.

Les utilisateurs suivants ont été créés :

```
dam@bpr.com - SomeSecurePassword123!

max@bpr.com - SomeSecurePassword123!

seb@bpr.com - SomeSecurePassword123!

cem@bpr.com - SomeSecurePassword123!

ced@bpr.com - SomeSecurePassword123!

bru@bpr.com - SomeSecurePassword123!
```

Annexe 5

Machine virtuelle BPR

Projet de semestre 5

La blockchain pour réaliser du recommandé

Machine virtuelle BPR

Le fichier « bpr.ova » permet de créer une machine virtuelle contenant une installation de notre proof of concept.

Les information de login pour la VM sont → user:pass

Pour lancer la blockchain, il faut exécuter le script « ~/bpr/init.sh »

Pour lancer le back-end, il faut exécuter le script « ~/bpr/backend/init.sh »

Pour lancer le front-end, il faut exécuter le script « ~/bpr/frontend/init.sh »

Se référer à la doc d'installation pour plus d'informations sur ces commandes.

Le front-end est accessible via « http://<VM_IP_ADDR>:8888 »

Le back-end est accessible via « http://<VM_IP_ADDR>:5000 » ou « https://<VM_IP_ADDR>:5001 »

Le serveur REST de Composer est accessible via http://<VM_IP_ADDR>:8080

Les utilisateurs suivants ont été créés :

```
dam@bpr.com - SomeSecurePassword123!

max@bpr.com - SomeSecurePassword123!

seb@bpr.com - SomeSecurePassword123!

cem@bpr.com - SomeSecurePassword123!

ced@bpr.com - SomeSecurePassword123!

bru@bpr.com - SomeSecurePassword123!
```