Mémoire d'ingénieur

Implémentation d'un service de liste de confiance globale basé sur la blockchain

Yoann Raucoules

Année 2016–2017

Stage de fin d'études réalisé dans l'entreprise ARHS Spikeseed

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Maître de stage : Vincent Bouckaert

Encadrant universitaire : Olivier Festor

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Raucoules, Yoann

Élève-ingénieur(e) régulièrement inscrit(e) en 3e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 1205028998

Année universitaire : 2016–2017

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Implémentation d'un service de liste de confiance globale basé sur la blockchain

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Luxembourg, le 3 septembre 2017

Signature :

Mémoire d'ingénieur

Implémentation d'un service de liste de confiance globale basé sur la blockchain

Yoann Raucoules

Année 2016–2017

Stage de fin d'études réalisé dans l'entreprise ARHS Spikeseed

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Yoann Raucoules
6, rue du général Frère
57070, METZ
+33 (0)6 77 48 04 38
yoann.raucoules@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

ARHS Spikeseed
2B, rue Nicolas Bové
1253, LUXEMBOURG
+352 26 11 02 1

Maître de stage : Vincent Bouckaert

Encadrant universitaire : Olivier Festor

Remerciements

Je tiens tout d'abord à remercier Madame Carlyne BALCZESAK, Monsieur Sébastien BRAUN et Monsieur Christophe GROSJEAN qui m'ont permis d'intégrer les équipes d'Arŋs pendant ces six mois de stage.

Je tiens également à remercier et à témoigner toute ma reconnaissance à mon maître de stage Monsieur Vincent BOUCKAERT pour cette expérience enrichissante et pleine d'intérêt, pour le temps qu'il m'a consacré tout au long de ce stage en répondant à mes interrogations.

Je souhaite aussi remercier Thomas LOISE et Francisco ROCHA qui ont contribué à ce projet et qui m'ont notamment aidé pour la rédaction de ce mémoire.

J'adresse aussi mes remerciements à l'ensemble des collaborateurs d'Arŋs qui par leur accueil et leur sympathie m'ont permis d'évoluer en toute sérénité au sein de l'entreprise.

Enfin, je tiens à remercier mon encadrant universitaire Monsieur Olivier FESTOR pour son intérêt porté au projet, son implication, son suivi tout au long de ce stage ainsi que pour sa visite dans les locaux de l'entreprise.

Avant-propos

Ce mémoire résulte d'un stage de fin d'études qui s'est déroulé du 3 avril 2017 au 30 septembre 2017 au sein de l'entreprise Arηs Spikeseed située au Luxembourg. Ce stage vient clôturer et valider la formation d'ingénieur du numérique de l'école TELECOM Nancy que j'ai débuté en septembre 2014. Cette formation qui s'est étendue sur une période de trois ans m'a permis d'acquérir de nombreuses compétences dans les domaines de l'informatique, des mathématiques, du management, de la gestion de projet, de la communication, de l'économie, du droit et des langues. J'ai choisi de me spécialiser en Ingénierie Logicielle au cours du cursus de par ma passion pour la programmation et l'architecture logicielle qui s'est révélée lorsque je me suis initié à l'informatique lors de mon stage de découverte professionnelle réalisé en classe de troisième.

Au cours de ce stage de fin d'études, j'ai eu le plaisir de travailler sur une technologie à laquelle je m'intéresse depuis deux ans, la blockchain. Dans le cadre d'un projet proposé par la Commission Européenne, nommé FutureTrust, j'ai pu concevoir et implémenter un service de liste de confiance globale basé sur la blockchain. Mes tâches ont été de me familiariser avec les principes de la blockchain et les concepts de cryptographie afin de les mettre en application dans le projet, d'effectuer une analyse des solutions de blockchain existantes afin de réaliser des choix d'implémentation, de concevoir l'architecture du service de liste de confiance globale, d'implémenter la solution conçue et de documenter tous les aspects techniques et fonctionnels de la solution implémentée.

Dans ce mémoire est présenté le résultat du stage de fin d'études et est mis en avant l'utilisation de la blockchain dans le cadre d'un projet de confiance numérique d'échelle mondiale. L'intérêt de ce document est dans un premier temps de détailler les tâches réalisées au cours du stage et dans un second temps de montrer qu'il est possible d'élargir le champ d'application de la technologie blockchain et des différents aspects qui la composent.

Table des matières

1 Introduction

La technologie blockchain s'est popularisée ces dernières années grâce à l'expansion de la crypto-monnaie [1] Bitcoin [2] [13] à travers le monde. En effet, cette technologie a bouleversé aussi bien le domaine de l'informatique que le domaine de la finance. L'investissement autour de la blockchain a mené à un engouement général pour ce concept. Le Bitcoin a réussi à remettre en cause des acteurs majeurs de notre société tels que les banques ou les géants du Web, en sécurisant des échanges d'actifs sans organe central de contrôle. La révolution qu'il a engendré amène aujourd'hui les gouvernements et autres organisations publiques à réfléchir sur la régulation de la technologie et des crypto-monnaies naissantes. Depuis son lancement en 2009, la blockchain n'a cessé d'évoluer et d'étendre son champ d'application. Bien que conçue à l'origine pour le transfert de crypto-monnaie, les avantages qu'elle apporte permettent d'imaginer de multiples cas d'utilisation qui dépassent son cadre initial. À l'heure où l'ubérisation [3] de notre société est en marche, la technologie blockchain amène une approche nouvelle qui permet de se détacher de tout organe central ou tierce partie. La blockchain ira-t-elle jusqu'à ubériser [4] Uber [5] ?

Dans ce contexte, un stage ingénieur a été réalisé sur une période de six mois au sein de la société Arŋs Spikeseed située au Luxembourg. Le stage a été réalisé dans les locaux de l'entreprise et la langue officielle du projet pour les communications avec les autres membres du consortium (mails, documents, conférences téléphoniques) a été l'anglais. La langue utilisée au sein de l'équipe a aussi été l'anglais, ceci étant dû au caractère multinational de celle-ci. Les documents produits dans le cadre du projet ont été rédigés en anglais et sont présentés en langue originale dans ce mémoire. Ce stage de fin d'études a eu pour objectif d'intégrer la technologie blockchain au sein d'un processus de gestion de listes de services de confiance dans le cadre de la mise en place d'un nouveau règlement européen. La finalité a été d'utiliser cette technologie afin de conserver des données publiques relatives à la confiance électronique de manière sécurisée et décentralisée en utilisant une blockchain en tant que registre. Cela a pour but d'assurer la disponibilité et l'intégrité des informations, puisque les données sont distribuées à travers les nœuds d'un réseau pair-à-pair et sécurisées à l'aide de transactions signées et vérifiées par une preuve mathématique.

Ce mémoire vise à montrer que le champ d'application de la technologie blockchain dépasse son cadre initial et que son utilisation permet de pallier aux problèmes d'architecture et de sécurité des modèles actuels. Dans un premier temps, le contexte du projet sera défini afin de présenter l'entreprise Arŋs Spikeseed, d'établir le contexte du projet et de détailler le fonctionnement de la technologie blockchain. Ensuite, la problématique sera exposée dans le but de comprendre les

1. La crypto-monnaie aussi appelée monnaie cryptographique est une monnaie électronique basé sur les principes de la cryptographie.
2. Bitcoin est une crypto-monnaie et un système de paiement pair-à-pair.
3. L'ubérisation est un phénomène économique désignant l'utilisation de services permettant aux professionnels et aux clients de se mettre en contact direct grâce à l'utilisation des nouvelles technologies.
4. Ubériser est le verbe issu du substantif ubérisation.
5. Uber est l'entreprise qui déclencha l'ubérisation.

différents besoins autour du service de liste de confiance globale et les limites de l'architecture actuelle. Ce chapitre détaillera aussi la méthodologie de gestion de projet ainsi que l'organisation du temps au cours du stage. À la suite de cela, sera établi un état de l'art afin d'analyser l'émergence de la blockchain et l'intérêt porté aux systèmes décentralisés. Dans ce chapitre seront aussi mis en avant la comparaison des outils existants et la justification des choix opérés durant le stage. Après cela, une analyse du problème sera développée en exposant les technologies utilisées, les acteurs du système, les cas d'utilisation de l'application ainsi que les modules, processus et modèle de données du service de liste de confiance globale. Puis, la réalisation de la solution sera détaillée en mettant en avant l'architecture mise en place, l'implémentation des différents modules ainsi que la présentation et la validation de la solution. Enfin, les résultats obtenus et les perspectives du projet seront présentés.

2 Présentation du contexte

Dans ce chapitre, nous présentons l'entreprise qui m'a accueilli dans le cadre de ce stage, le contexte dans lequel s'inscrit le projet réalisé ainsi qu'une introduction à la technologie blockchain.

2.1 L'entreprise Arηs Spikeseed

Arηs Spikeseed est une entité du groupe Arηs qui est une entreprise de services du numérique (ESN) fondée en 2003 par Jourdan Serderidis. Le groupe est divisé en sociétés réparties au Luxembourg, en Belgique, en Grèce et depuis cette année en Italie. Le groupe Arηs possèdent cinq axes de compétences qui sont présentés dans la Figure 2.1.

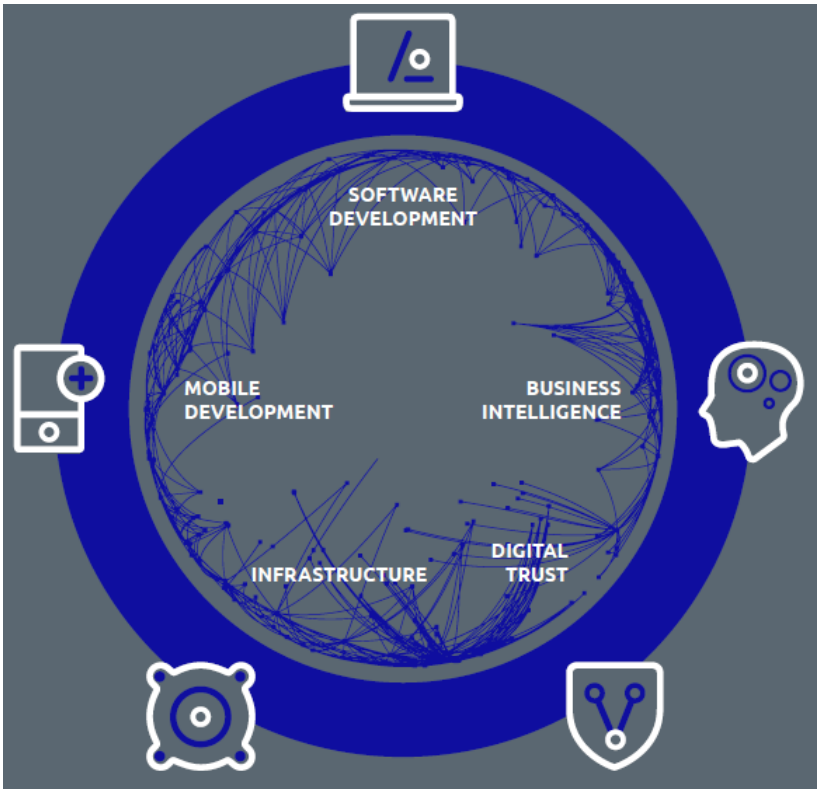


FIGURE 2.1 – Axes de compétences du groupe Arηs. Illustration extraite de *Annual Report 2016* [2]

Comme toutes les autres entités du groupe, Arηs Spikeseed vise à délivrer des solutions numériques complexes. Elle a la particularité de réaliser principalement des projets de recherche et développement en s'appuyant sur les pratiques agiles et des technologies de pointe. De plus, Arηs Spikeseed est compétente afin de mettre en œuvre des solutions liées à la confiance numérique,

des systèmes engageant des masses de données grâce à des technologies innovantes et efficaces comme le Web sémantique ou la Business intelligence ainsi que des applications destinées aux mobiles et aux objets connectés.

2.2 Contexte du projet

Les architectures logicielles évoluent en suivant les innovations technologiques. Le domaine de la recherche apporte de nouvelles technologies et des améliorations aux concepts existants à une vitesse exponentielle. L'univers technologique poussé par l'innovation oblige les acteurs du numérique à s'adapter en permanence aux changements. La technologie blockchain s'inscrit dans ces innovations récentes issues de la recherche. Elle amène une nouvelle vision d'un Internet décentralisé, sans organe central de contrôle, qui va probablement révolutionner la conception des systèmes d'information dans les prochaines années.

Dans le cadre de la mise en place d'un règlement de l'Union Européenne (UE) sur l'identification électronique et les services de confiance pour les transactions électroniques sécurisées au sein de l'UE (eIDAS), la Commission Européenne (CE) a proposé un projet qui a pour visée de supporter la mise en œuvre technique de ce règlement européen. Le règlement eIDAS [1] a pour but d'établir un cadre d'interopérabilité pour les systèmes mis en place au sein de l'UE afin de promouvoir le développement d'un marché de la confiance numérique, et d'instaurer un mécanisme de reconnaissance mutuelle des moyens d'identification électronique des États membres sur l'ensemble des services fournit par d'autres États membres.

Ce projet de recherche et développement appelé FutureTrust rassemble un consortium de seize partenaires, dont Arηs Spikeseed. Le projet FutureTrust répondra au besoin de solutions globales et interopérables, en fournissant des logiciels libres qui faciliteront l'utilisation de l'identification et de la signature électronique. Il vise à étendre l'infrastructure de la liste européenne de services de confiance existante vers une liste mondiale des services de confiance nommée Global Trust Service Status List (gTSL), à développer un service de validation ainsi qu'un service d'archivage pour les signatures et les sceaux électroniques, et à fournir des composants pour les certificats qualifiés et pour la création de signatures et de sceaux dans un environnement mobile.

Dans ce contexte, l'intégration de la blockchain a été proposée dans le cadre du projet FutureTrust et plus particulièrement dans le module de gTSL. Les autres modules du projet n'entrent pas dans le cadre du stage et ne sont pas détaillés dans ce document.

2.3 Présentation de la technologie blockchain

Une blockchain est une technologie basée sur l'échange d'actifs numériques, réalisé grâce à des transactions signées, et agit comme un registre publique distribué où toutes les transactions émises sont répertoriées. Elle repose sur des principes de cryptographie afin d'assurer l'intégrité de ces transactions et sur un protocole décentralisé, dit "peer-to-peer". Ce protocole permet à la blockchain d'avoir une disponibilité maximale et d'établir un consensus entre les participants du réseau afin de protéger les transactions contre les falsifications. La Figure 2.2 représente le processus d'émission et de validation d'une transaction sur la blockchain.

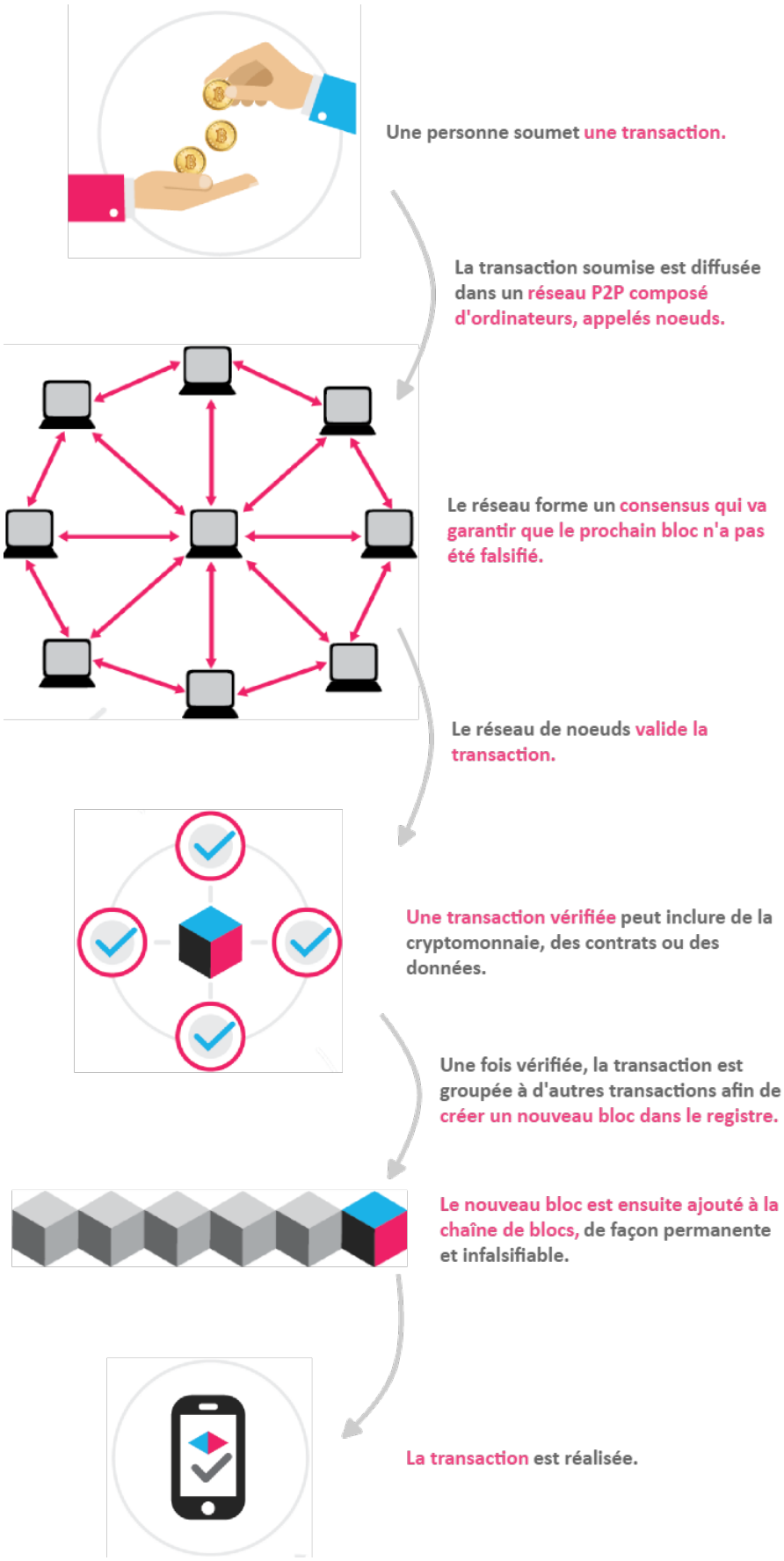


FIGURE 2.2 – Processus de création et de validation d'une transaction sur la blockchain. Illustration adaptée de *What is Blockchain Technology ?* [6]

Dans la Figure 2.2, nous remarquons que les blocs sont agencés dans un ordre linéaire, à la manière de maillons dans une chaîne, et contiennent une référence au bloc précédent, ainsi qu'un enregistrement des transactions. La preuve de travail est le traitement nécessaire pour générer un nouveau bloc basé sur les nouvelles transactions diffusées sur le réseau. Les blocs de transaction sont créés par un processus appelé le minage [1], qui est conçu pour être coûteux en temps et en énergie ainsi qu'être complexe à réaliser, et s'appuie sur un consensus pour ajuster la difficulté de créer de nouveaux blocs. Le minage est aussi un moyen de sécuriser le réseau en créant, vérifiant, publiant et propageant les blocs dans la blockchain.

2.3.1 Introduction aux Smart Contracts

En 1994, Nick Szabo, chercheur juridique et cryptographe, s'est rendu compte que le registre décentralisé pouvait être utilisé pour des smart contracts (contrats intelligents), autrement appelés contrats auto-exécutés, contrats blockchain ou contrats numériques. Les contrats peuvent être convertis en code informatique, stockés et répliqués sur le système. Les smart contracts permettent d'échanger de l'argent, des biens, des parts ou n'importe quel actif de manière transparente, sans conflits et en évitant tout service intermédiaire. La Figure 2.3 représente l'établissement d'un smart-contract entre deux parties, sans service intermédiaire. Dans ce mémoire, nous utilisons le terme de smart contract pour désigner un contrat intelligent car ce terme est le plus répandu.

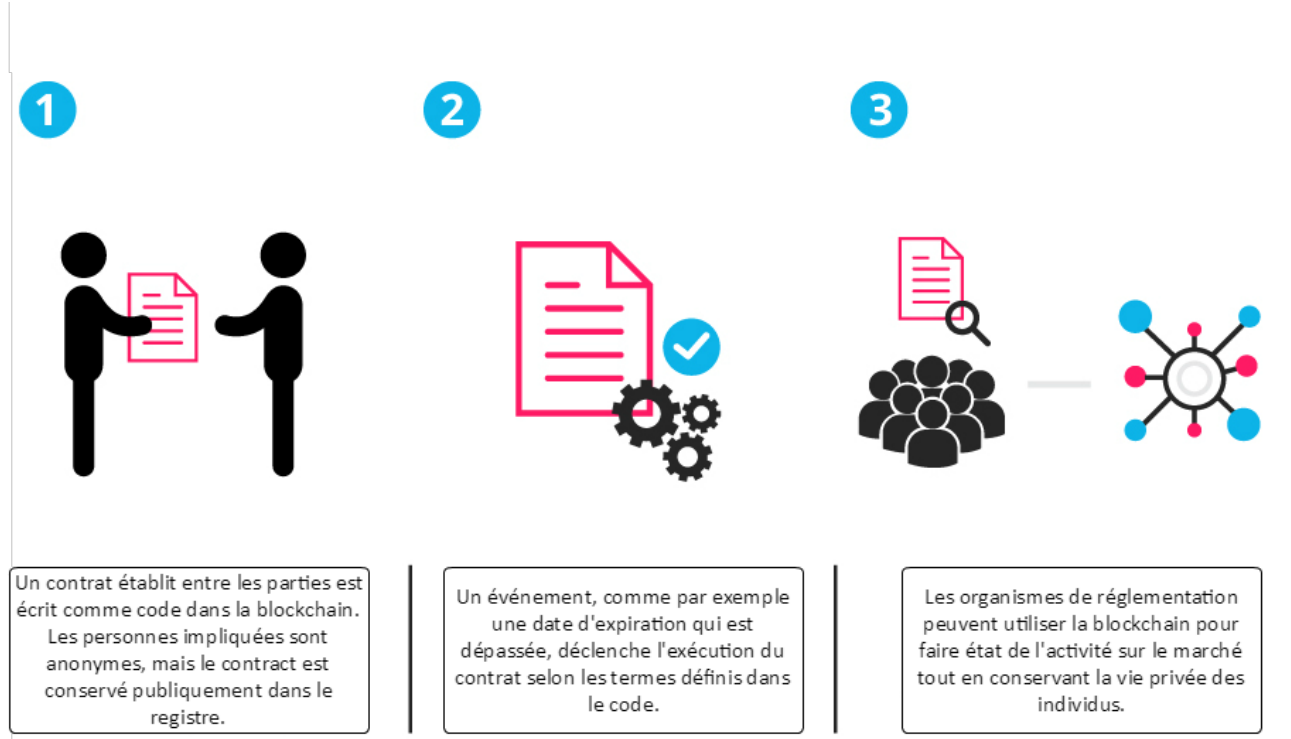


FIGURE 2.3 – Établissement d'un smart-contract. Illustration adaptée de *Smart Contracts : The Blockchain Technology That Will Replace Lawyers* [5]

1. mining en anglais.

Les smart-contracts apportent de nombreux avantages pour leurs utilisateurs :

- l'autonomie, qui signifie que l'accord engendré par le contrat n'a pas besoin d'être confirmé par un intermédiaire tierce comme un notaire ou un avocat ; Cela empêche aussi toute manipulation par un tiers sur l'un des parties signataires du contrat puisque l"exécution du contrat est géré automatiquement par le réseau, ce qui empêche tout individu biaisé d'avoir une influence sur le contrat ;
- la confiance, puisque les documents sont encodés dans le registre distribué, il n'y a aucun moyen qu'ils soient perdus ;
- la sauvegarde, toutes les informations sont stockées dans la blockchain et sont répliquées à travers de multiples nœuds du réseau qui les maintiennent ;
- la sécurité, les concepts de cryptographie utilisés permettent de conserver les contrats en sécurité, ils ne peuvent être modifiés ou usurpés ;
- la vitesse, les smart-contracts permettent d'accélérer les procédures, en effet toutes les tâches sont automatisées donc aucun tiers ne ralentit le processus ;
- des économies, en effet se détacher des intermédiaires permet de réduire considérablement les charges administratives engendrées par la plupart des contrats classiques ;
- la précision, puisque le contrat est automatisé et exécuté par une machine, si l'on admet que le contrat a été codé correctement, alors toute erreur pouvant être introduite lors d'une rédaction manuelle ne peut être présente dans un smart-contract.

3 Présentation détaillée de la problématique

Dans ce chapitre, nous présentons les besoins liés au service de liste de confiance globale ainsi que les limites de l'architecture actuelle. Ce chapitre introduit aussi les concepts relatifs à la gestion de projet, plus précisément la méthodologie utilisée et l'organisation du temps au cours du stage.

3.1 Description des besoins du service de liste de confiance globale

Les États membres de l'UE et d'autres pays européens maintiennent généralement des listes d'informations sur des autorités de certification, ainsi que des listes d'informations sur des fournisseurs de services, désignés Trust Service Providers (TSP), dans un ou plusieurs registres à l'échelle nationale.

La liste de confiance des États membres de l'UE comprend des informations relatives aux TSPs qualifiés qui sont supervisés par l'État membre compétent, ainsi que des informations relatives aux services de confiance, désignés Trust Services (TS), qu'ils fournissent, conformément aux dispositions prévues par le règlement eIDAS. En effet, les listes de confiance sont des éléments essentiels dans la mise en place de la confiance numérique pour les opérateurs du marché électronique, en permettant aux utilisateurs de déterminer le statut qualifié des TSPs et de leurs TSs.

En vertu du règlement eIDAS, les listes nationales de confiance ont un effet constitutif. En d'autres termes, un fournisseur ou un service ne sera qualifié que s'il apparaît dans les listes de confiance. Par conséquent, les utilisateurs (citoyens, entreprises ou administrations publiques) bénéficieront de l'effet juridique associé à un service de confiance qualifié uniquement si ce dernier est répertorié (comme qualifié) dans les listes de confiance. Les États membres peuvent inclure dans les listes de confiance des informations sur les fournisseurs de services de confiance non qualifiés et sur d'autres services de confiance définis au niveau national.

La structure d'une liste de confiance est présentée dans la Figure 3.1.

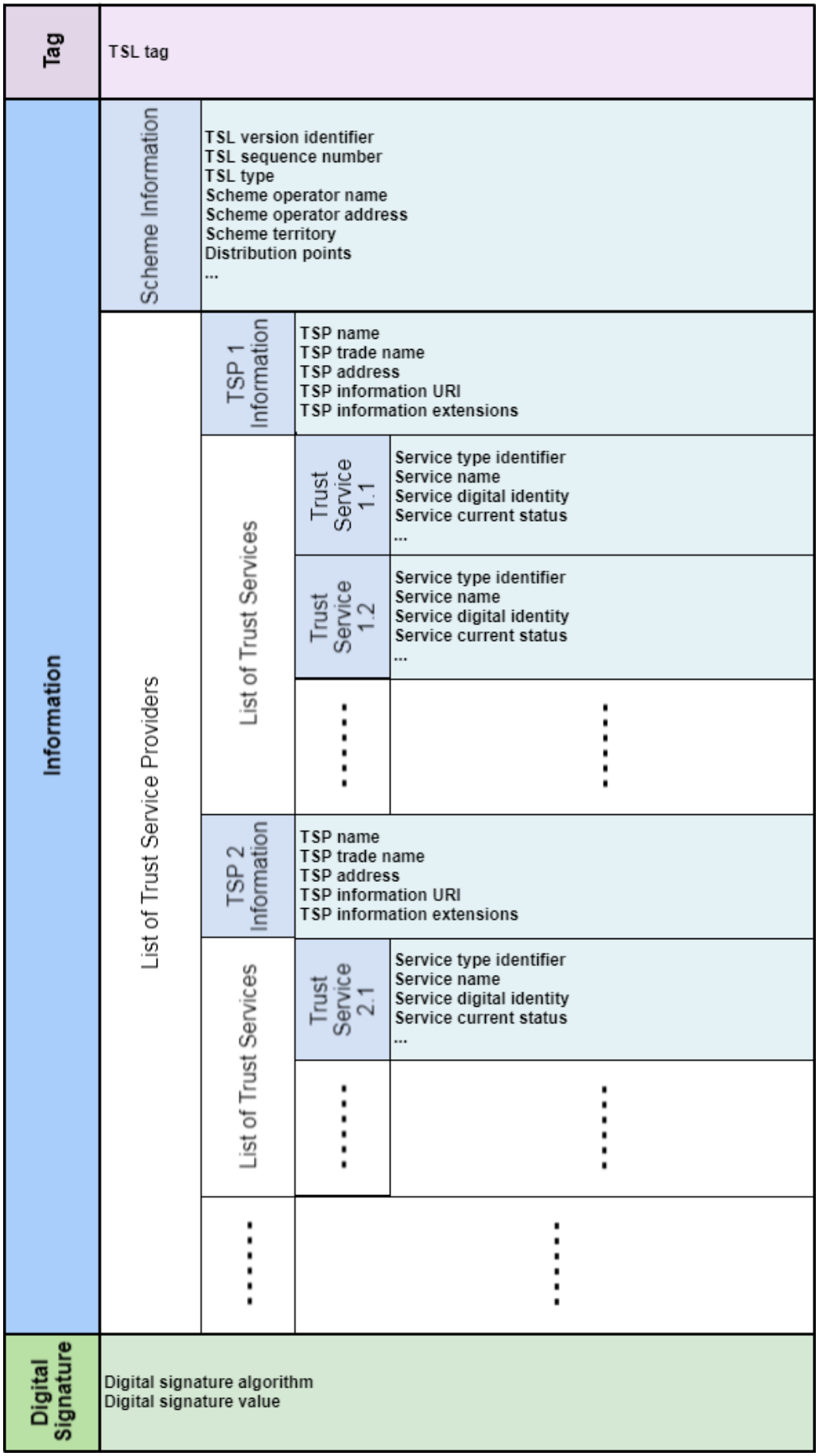


FIGURE 3.1 – Structure d'une liste de confiance

Dans la Figure 3.1, on distingue qu'une liste de confiance peut être décomposée en trois sections.

Tag

La section *Tag*, et plus particulièrement son attribut *TSL Tag*, est une URI [1] qui permet d'indiquer le standard sur lequel s'appuie la liste de confiance. Actuellement, le seul standard existant est ETSI TS 119 612 [11]. Il est possible qu'un nouveau standard soit défini dans le futur, cette section permettra donc d'indiquer le standard sur lequel la liste de confiance est basé.

Information

La section *Information* peut-être divisée en deux parties. La première partie, nommée *Scheme Information*, répertorie toutes les informations relatives à la liste de confiance comme par exemple sa version, le nom et l'adresse de l'opérateur de la liste ou encore le pays pour lequel la liste est définie. Il est important de noter que dans la Figure 3.1, la liste des informations citées n'est pas exhaustive. La seconde partie est la liste des TSPs qui répertorie l'ensemble des fournisseurs approuvés par l'État membre. Pour chacun des TSPs, on retrouve ses informations ainsi que la liste des services de confiances qu'il fournit.

Digital Signature

La section *Digital Signature* permet de vérifier l'authenticité et l'intégrité de la liste de confiance. En effet, chaque liste doit être signée par un opérateur désigné et défini dans la partie *Scheme Information*. Dans cette section doit être indiquée la signature de l'opérateur ainsi que l'algorithme de génération de celle-ci.

3.1.1 Besoins généraux

Résumé des objectifs du projet

L'intérêt du service de gTSL est de favoriser l'établissement de relations de confiance entre les opérateurs du marché en Europe et au-delà. Cette liste a pour but de répertorier les TSPs, ayant un statut qualifié ou non. On entend par statut qualifié que le TSP ait été accrédité par un organisme compétent au sein de l'État membre dans lequel le TSP est déclaré. Le service permet aux utilisateurs finaux de vérifier le statut de ces TSPs et d'accéder à l'ensemble des informations concernant les services de confiance. À ce titre, elle étend le schéma actuel de la liste des services de confiance, dont la portée est uniquement européenne. De plus, cette réorganisation de l'architecture vise à gérer la gTSL de manière décentralisée dans le but d'en améliorer sa résilience.

Parties prenantes

Les acteurs principaux de la gTSL sont :

1. Une URI (acronyme anglais de Uniform Resource Identifier) est une chaîne de caractères identifiant une ressource.

— les États membres de l'UE, qui doivent établir, maintenir et publier les listes de confiance, incluant les informations relatives aux TSPs déclarés au sein de leur État ;
— les fournisseurs de services de confiance, qui sont destinés à s'appuyer sur le service de gTSL dans lequel sont publiés leur statut qualifié et leurs informations publiques ;
— les opérateurs de listes de confiance ne faisant pas partie d'un État membre de l'UE, qui souhaitent intégrer leur liste dans la gTSL ;
— les citoyens de l'UE et non UE, qui sont destinés à utiliser le service afin d'accéder aux statuts et aux informations des différents TSPs répertoriés dans la gTSL.

3.1.2 Besoins du système

Objectif du système

En s'appuyant sur la norme régissant les listes de confiance définie dans ETSI TS 119 612 [11], la gTSL vise à résoudre les imperfections actuelles du schéma de liste de confiance, énoncées dans la Section 3.2.

À l'heure actuelle, la Commission européenne publie une liste signée de pointeurs, nommée European List of the Lists (LoTL), dans laquelle chaque pointeur désigne un point de distribution pour une liste nationale de TSPs. Ces listes nationales contiennent des informations sur les TSPs qualifiés et non qualifiés ainsi que sur les TSs qualifiés ou non qualifiés qu'ils proposent.

À ce titre, la gTSL fournira les fonctions nécessaires à la création, à la mise à jour et à la distribution des TSPs et des informations concernant leurs TSs. Le but principal de la gTSL est de s'appuyer sur le modèle de distribution centralisé actuel et de l'adapter à un nouveau modèle décentralisé. L'émergence récente de la technologie blockchain ainsi que l'engouement général de la part de la communauté open-source autour de la décentralisations qu'elle rend possible nous apporte un contexte d'utilisation idéal dans notre objectif de décentralisation.

Présentation du système

Afin d'atteindre ses objectifs, la gTSL s'appuie sur quatre principaux composants :
— le *Global Trust Service Responder*[2] qui permet aux applications externes et aux utilisateurs d'interroger la gTSL afin de récupérer les informations relatives aux TSPs ;
— l'interface d'administration qui a pour objectif de fournir une interface graphique ergonomique aux opérateurs afin qu'ils puissent facilement procéder à des modifications sur les listes de confiance ;
— le *Global Trust Service Lifecycle Manager*[3] qui a pour vocation de faciliter la gestion des services de confiance à travers une interface permettant la création, la mise à jour et la distribution des informations relatives aux statuts de confiance ;
— le *Ledger Manager* qui a pour rôle de manipuler et de stocker les données.

Ces composants et leurs interactions sont illustrés dans la Figure 3.2.

2. en français, Répondeur (dans le sens où il répond aux requêtes des utilisateurs).
3. en français, Gestionnaire du cycle de vie.

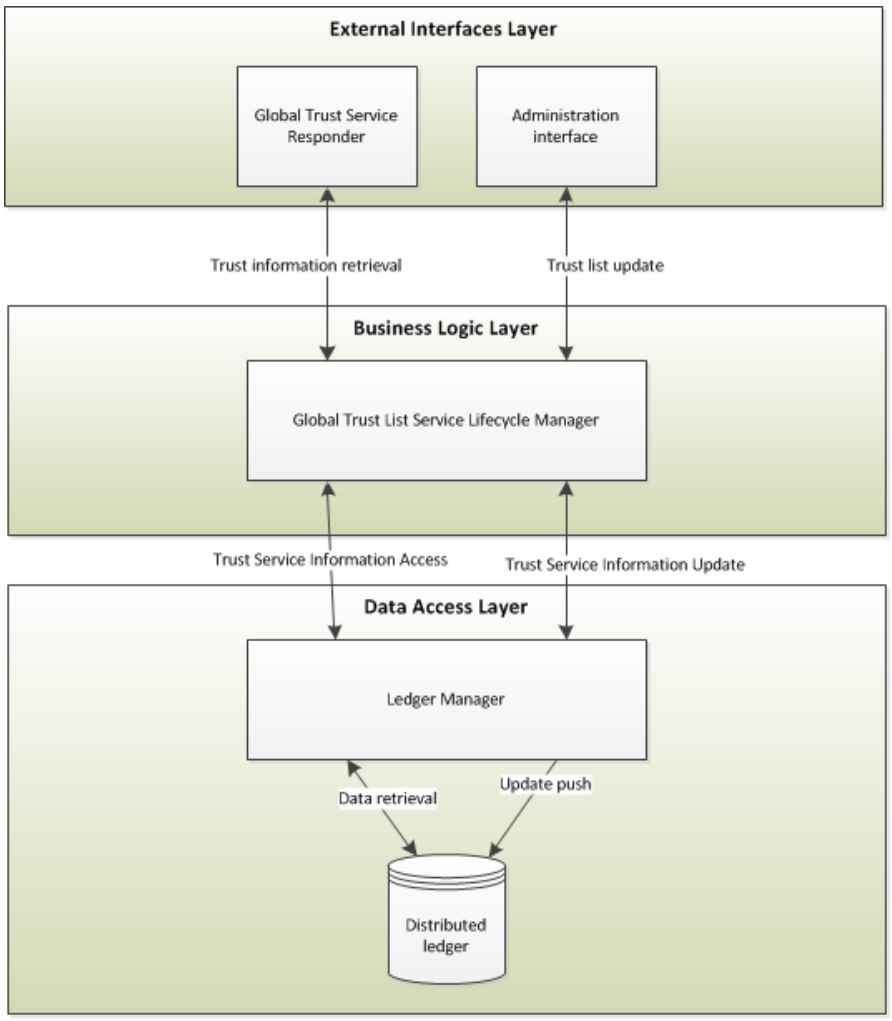


FIGURE 3.2 – Architecture 3 tiers de la gTSL. Illustration extraite de *Design documentation* [7]

D'un point de vue architectural, la gTSL s'appuie sur une architecture à trois couches :

- la couche de services externes qui expose les interfaces externes du système, i.e. le *Global Trust Service Responder* et l'interface d'administration ;
- la couche métier qui est composée du *Global Trust List Service Lifecycle Manager* ;
- la couche de données qui correspond aux interfaces et aux composants qui permettent de connecter la gTSL à une solution de stockage de données.

Caractéristiques des utilisateurs

Deux types différents d'utilisateurs ont été identifiés concernant la gTSL :

- les utilisateurs d'administration, i.e. les administrateurs des listes, qui peuvent agir au nom d'un État membre de l'UE et qui sont chargés de la maintenance quotidienne et de la gestion des listes de confiance ;
- les utilisateurs externes, i.e. les personnes et les applications externes qui souhaitent obtenir des informations concernant les statuts de confiance pour un TSP, un TS ou un État membre donné.

Ces utilisateurs ont accès au système à travers des interfaces dédiées :

- les utilisateurs d'administration ont accès à une plateforme de gestion de la gTSL, qui expose sans ambiguïté les différentes fonctionnalités d'administration auxquelles ces uti-

lisateurs doivent avoir accès ;
— les utilisateurs externes ont accès à la fois à une interface graphique et à une seconde interface de web services [4], qui permet la récupération d'informations concernant les statuts des services de confiance sur base de certificats électroniques fournis par l'utilisateur ainsi que des informations générales concernant les TSPs.

Besoins fonctionnelles

La gTSL doit supporter l'internationalisation (non-UE) prévue dans le règlement eIDAS. À ce titre, la gTSL doit permettre l'ajout de TSPs déclarés dans un pays qui n'est pas membre de l'UE, qu'ils soient qualifiés ou non.

Besoins d'utilisation

La gTSL doit offrir une interface permettant la récupération ainsi que la publication d'informations relatives aux TSPs. L'objectif minimum à remplir pour assurer la conformité avec le standard ETSI TS 119 612 [11] est que la gTSL soit disponible via le protocole HTTP.

Besoins de performance

La gTSL doit apporter un stockage interne efficace pour stocker les informations concernant les TSPs et doit être hautement évolutif afin de gérer efficacement de nombreuses quantités de demandes parallèles.

Interfaces du système

La gTSL doit exposer, grâce à une interface de web services, les fonctionnalités permettant la récupération d'informations concernant les statuts des services de confiance.

Interfaces utilisateur

Les fonctionnalités de gestion de la gTSL doivent être fournies à travers une interface web cohérente et intuitive, permettant aux utilisateurs de l'utiliser sans ambiguïté. Les interfaces utilisateur doivent rester cohérentes avec les interfaces utilisateur de l'application actuelle tout en ne montrant aucune ambiguïté en termes de hiérarchie visuelle et de contenu.

Fiabilité du système

Le système doit être disponible sur une base de 24 heures par jour et 7 jours par semaine. Plus particulièrement, le standard ETSI TS 119 612 [11] impose que la récupération des informations concernant les listes de confiance soit disponible sur une base de 24 heures par jour et 7 jours par semaine, avec une disponibilité annuelle minimum de 99.9%.

4. Un web service correspond à l'implémentation d'une ressource identifiée par une URL.

Sécurité du système

En raison de la nature sensible des données gérées par la gTSL, et de la haute disponibilité requise, les besoins en terme de sécurité doivent garantir que ces données ne peuvent être et ne sont pas compromises. De plus, il est primordial que la gestion des services de confiance et des fournisseurs de services de confiance soit limitée aux personnes autorisées. En effet, la gTSL ne doit pas permettre à des personnes non autorisées de créer, modifier ou supprimer des informations relatives aux listes de confiance et doit assurer l'intégrité des données qu'elle traite.

3.1.3 Besoins logicielles

Conformité au standard

La gTSL doit respecter la syntaxe et la sémantique d'une liste de confiance définies dans le standard ETSI TS 119 612 [11].

Rétrocompatibilité

La gTSL doit pouvoir s'intégrer au schéma existant basé sur la LoTL. Cela signifie qu'elle doit être rétrocompatible avec l'ensemble des listes de confiance actuellement référencées dans la LoTL.

3.2 Limites de l'architecture actuelle

Dans le modèle actuel, chaque modification apportée au contenu d'une liste nationale induisent la nécessité de republier l'entièreté de cette liste. De plus, toutes modifications apportées sur l'URL [5] à laquelle la liste est distribuée ou sur le certificat utilisé pour signer la liste, induisent la nécessité de republier à la fois la liste nationale et la liste européenne.

Le caractère centralisé du système de distribution des listes de confiance actuel contient des problèmes éventuels qui doivent être résolus dans le cadre de la globalisation des listes de services de confiance :

- les listes de confiance des États membres sont uniquement récupérables après inspection préalable de la LoTL, le schéma actuel est donc sujet à un point individuel de défaillance [6] ;
- chaque État membre maintient les données relatives à sa liste de confiance, cela signifie que l'arrêt du nœud de distribution d'un État membre rend ses données non consultables ;
- l'architecture existante est exposée à un problème de résilience puisqu'elle nécessite que l'ensemble des nœuds de distribution des États membres soient actifs afin que la liste globale soit considérée complète et donc fiable ;
- l'intégrité des données peut être compromise, car si les données d'un État membre sont corrompues localement sur son nœud de distribution, alors l'intégrité globale est compromise puisque la confiance accordée peut être brisée par un seul nœud défaillant ;

5. Une URL (acronyme anglais de Uniform Resource Locator) est couramment appelé adresse web.

6. Un point individuel de défaillance (single point of failure ou SPOF en anglais) est un point d'un système informatique dont le reste du système est dépendant et dont une panne entraîne l'arrêt complet du système.

— des problèmes de performance et de latence peuvent être rencontrés puisqu'il est nécessaire de télécharger et valider l'ensemble des informations qui sont réparties sur différents points de distribution ;
— le schéma actuel ne conserve pas l'historique des modifications, c'est-à-dire qu'une nouvelle publication d'une liste remplace totalement la précédente, ce qui ne permet pas de conserver une trace des modifications mises en œuvre entre les versions, ni de revenir à une version antérieure en cas de problème.

L'objectif de la gTSL est d'effectuer une refonte de l'architecture actuelle qui a montré ses limites en y apportant des technologies innovantes. Pour cela, il est nécessaire d'adopter un modèle décentralisé et distribué qui permet de résoudre les problèmes de résilience et de point individuel de défaillance sus-cités. Par ailleurs, la technologie blockchain apporte des avantages permettant d'assurer l'intégrité des données ainsi que la sécurité du système.

3.3 Gestion de projet

Cette section a pour objectif d'introduire la méthode Agile qui a été utilisée au cours de ce stage, et d'exposer la répartition du travail réalisé, à l'aide d'un diagramme de Gantt.

3.3.1 Méthode de gestion de projet

La méthode Agile a été utilisée au cours de ce projet. Plus particulièrement, l'équipe s'est appuyée sur le schéma Scrum, qui permet un cadre de travail itératif et incrémental où les tâches majeures sont décomposées en sous-tâches. La méthodologie Scrum est basée sur le découpage d'un projet en sprints [7], qui sont des cycles de livraison très courts. Un sprint peut s'étendre sur une durée de quelques heures à un mois. Dans notre cas, nous avons choisi une durée de deux semaines par sprint, qui est la durée préconisée dans la méthode. En début de sprint, une estimation de la durée de chaque tâche est effectuée. Ensuite une planification opérationnelle est réalisée. Un sprint se termine généralement par une démonstration du travail réalisé, suivie d'une rétrospective afin d'analyser le déroulement du sprint achevé dans le but d'améliorer les pratiques de l'équipe. Quotidiennement est organisé un scrum meeting [8], réunion courte et énergique, qui permet à l'équipe de discuter de l'avancée du sprint et de lever les points bloquants du projet. Afin de suivre la progression des objectifs au fur et à mesure de l'avancement du projet, nous avons utilisé l'outil JIRA [9], qui est un système de gestion de projets. Il a servi notamment à définir les différentes tâches du projet et à répartir le travail entre les membres de l'équipe.

Les avantages majeures de Scrum

Scrum apporte des avantages qui sont définis comme étant les trois piliers de la méthodologie :
— la transparence, par l'utilisation d'un langage commun afin de permettre à tout un chacun d'obtenir rapidement une bonne compréhension du projet ;
— l'inspection, par l'analyse quotidienne du travail accompli et restant lors des sprints, afin de repérer tout indicateur indésirable ;

7. Un sprint est une période sur laquelle sont réalisées des tâches définies.
8. Un scrum meeting est communément appelé mêlée en français.
9. Voir https://fr.atlassian.com/software/jira

— l'adaptation, dans le cas d'une dérive après inspection, des ajustements doivent être effectués afin de minimiser les écarts de réalisation.

3.3.2 Organisation du temps

La Figure 3.3 présente un diagramme de Gantt qui expose de manière globale la répartition du travail réalisé. Ce diagramme est volontairement non détaillé puisque l'utilisation de la méthodologie Agile ne permet pas d'organiser à l'avance et avec précision la réalisation des tâches. Il est important de noter que le diagramme se termine à la date de fin du stage mais que la livraison du projet est prévue au 30 novembre 2017.

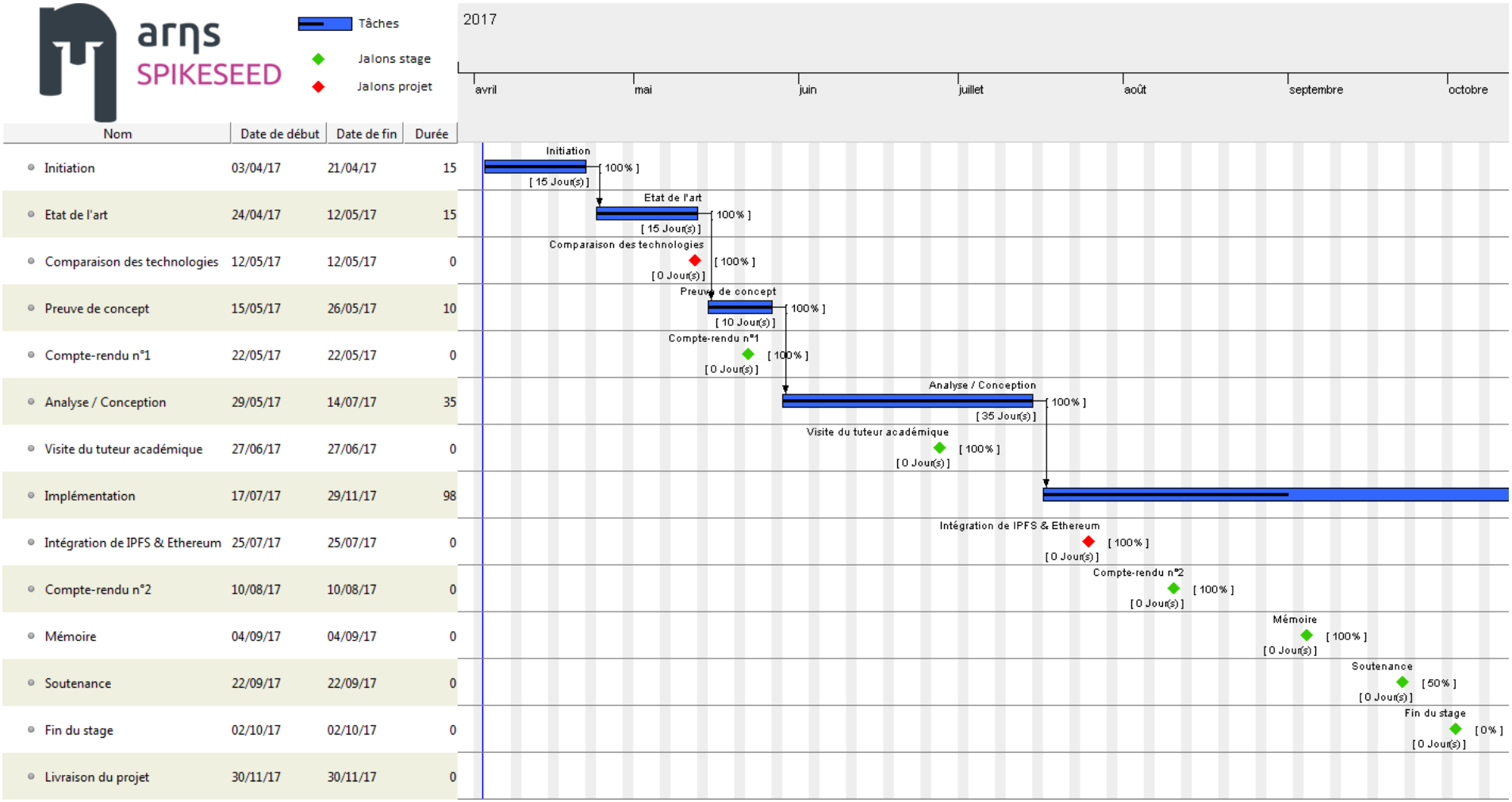


FIGURE 3.3 – Diagramme de Gantt

Le diagramme de Gantt en Figure 3.3 permet d'avoir un aperçu de l'organisation du stage. Le stage s'est déroulé sur une période de six mois, du 3 avril 2017 au 30 septembre 2017. Durant cette période, j'ai été amené à réaliser différentes tâches qui sont détaillées dans la suite de ce mémoire. D'un point de vue générale, le stage a été décomposée en cinq parties majeures qui sont décrites ci-après.

Initiation

La phase d'initiation a commencé dès le début du stage et a duré environ trois semaines. Elle peut être décomposée en deux sous-parties. Dans un premier temps, j'ai dû me familiariser avec le projet. Pour réaliser cela, j'ai eu accès au document de conception de la gTSL [7] qui explique, d'une manière générale et sans précision de technologies, l'architecture à mettre en place ainsi que les cas d'utilisation à implémenter dans le cadre du projet FutureTrust. J'ai pris connaissance du standard ETSI TS 119 612 [11] qui détaille le format à respecter dans le cadre des listes de confiance. Dans un second temps, j'ai effectué des recherches sur les notions de cryptographie, de signature électronique et de blockchain afin de les mettre en œuvre dans le projet. Cette seconde partie a été la préparation du travail suivant qui est l'état de l'art.

État de l'art

La seconde phase, ayant pour objectif d'établir un état de l'art, a suivi la phase d'initiation et s'est étendue sur trois semaines. L'état de l'art est un processus qui vise à explorer l'état des connaissances et technologies actuelles dans un domaine spécifique. Pour notre projet, l'état de l'art a porté sur la blockchain ainsi que les systèmes décentralisés. Le but de cette phase a été d'explorer le plus largement possible les technologies pouvant être utilisées dans le projet afin de les comparer pour choisir par la suite la solution la plus adaptée à nos besoins. Cette étape a donné lieu à un livrable qui a été inclus dans le document de conception de la gTSL comme le montre le jalon "Comparaison des technologies" dans la Figure 3.3. Ce document est disponible en Annexe E. L'état de l'art est détaillé dans la Chapitre 4.

Preuve de concept

À la suite de l'état de l'art, un choix technologique a été effectué. L'étape suivante a donc consisté à prouver que les choix opérés correspondent aux besoins du projet. Pour cela, j'ai pu réaliser une preuve de concept sur une durée de quinze jours. Cette phase a ensuite donné lieu à une démonstration à l'équipe afin de valider de manière définitive les choix technologiques. Le détail de la preuve de concept ainsi que la justification des choix opérés sont présentés dans la Chapitre 6.

Analyse / Conception

La phase d'analyse et conception a été précédée de la preuve de concept qui a permis de valider les technologies à utiliser pour le projet. Dans le cadre de cette phase, nous avons déterminé les modules à implémenter afin de répondre aux besoins exprimés dans le document de conception. L'analyse et la conception ont consisté à imaginer des potentielles solutions quant à la mise en place d'une architecture décentralisée basée sur une blockchain en étant conforme aux diffé-

rentes contraintes définies dans le document de conception et en s'appuyant sur les technologies choisies. Cette partie a duré un mois et demi, et est détaillée dans la Chapitre 5.

Implémentation

La dernière phase, et la plus conséquente, est l'implémentation. Elle consiste à développer la solution conçue et imaginée lors de la phase d'analyse et conception. Lors de la rédaction de ce mémoire, cette phase est en cours de réalisation. Cette phase est découpée en sprints d'une durée de deux semaines. L'implémentation est détaillée dans la Chapitre 6.

4 État de l'art

Dans le cadre du projet FutureTrust, il a été nécessaire d'effectuer un état de l'art afin d'avoir une vue globale de l'état actuelle de la technologie blockchain et des technologies distribuées et décentralisées comme les bases de données ou les systèmes de fichiers. Pour cela, il convient de décrire l'émergence de la blockchain ces dix dernières années, d'expliquer les avantages qu'apporte la décentralisation dans les systèmes actuels et d'établir une liste des technologies existantes. L'état de l'art a pour intérêt de soulever l'effervescence autour de ces technologies ainsi que d'opérer les meilleurs choix dans la réalisation du module de gTSL. Les solutions existantes présentées en Section 4.3 ont été envisagées afin de mettre en place un système permettant de conserver, gérer et récupérer les données de la gTSL de manière sécurisée et apportant une forte résilience.

4.1 L'émergence de la blockchain

Cette section a pour objectif de comprendre l'origine de la blockchain en détaillant les recherches antérieures à cette technologie, et d'analyser son état actuel et les améliorations qui peuvent être apportées dans le futur.

4.1.1 Historique

Le concept de monnaie numérique, ainsi que d'autres applications comme les registres de propriété, existent depuis des décennies. Les protocoles de paiement apparus dans les années 1980 et 1990, connus sous le nom de e-cash[9] et reposant sur des concepts cryptographiques, avaient pour but de fournir une monnaie avec un degré élevé de confidentialité. La mise en place de ces protocoles a largement échoué à cause de leur dépendance à un intermédiaire centralisé. En 1998, la technologie B-money[10] créé par Wei Dai voit le jour, et devient la première proposition introduisant l'idée de créer de l'argent en résolvant des puzzles informatiques en se basant sur un consensus décentralisé. Cependant, cette proposition a été peu détaillée sur la manière dont le consensus décentralisé pouvait effectivement être mis en œuvre. En 2005, Hal Finney a introduit le concept de preuves de travail réutilisables, un système basé sur des idées de b-money en les combinant aux puzzles Hashcash[3] difficiles à calculer créés par Adam Back, afin de créer un concept de crypto-monnaie. Mais encore une fois, cette proposition n'a pas su être exploitée. En 2009, une monnaie décentralisée a été pour la première fois mise en œuvre par Satoshi Nakamoto, combinant les différentes propositions établies pour la gestion de la propriété par l'utilisation de clés publiques avec un algorithme de consensus permettant de suivre l'identité du possesseur de la monnaie, appelé preuve de travail (Proof of Work). C'est la naissance du Bitcoin et plus généralement de la blockchain.

Le mécanisme derrière la preuve de travail a été une avancée car il résout simultanément deux

problèmes. Tout d'abord, il a fourni un algorithme de consensus simple et modérément efficace, permettant aux nœuds du réseau de convenir collectivement d'un ensemble de mises à jour du registre Bitcoin. Deuxièmement, il a fourni un mécanisme qui permet l'entrée libre d'un nouveau membre dans le processus de consensus, ce qui résout le problème engendré par l'influence d'un membre dans le consensus, tout en empêchant les attaques Sybil [1]. Dans la preuve de travail, le poids d'un nœud dans le processus de vote par consensus est directement proportionnel aux ressources de calcul que ce nœud apporte. Depuis cela, une approche alternative, appelée preuve d'enjeu (Proof of Stake), initiée par Peercoin[12] a été proposée, dans laquelle le poids d'un nœud est proportionnel à la quantité de crypto-monnaie qu'il détient et non plus aux ressources informatiques qu'il apporte.

4.1.2 État actuel et potentiel futur

Actuellement deux principaux protocoles publiques se sont démarqués. Le premier est bien entendu Bitcoin et le second est le récemment créé Ethereum. Malgré les solutions qu'ils apportent, les défis et les développements potentiels de la blockchain restent nombreux. Premièrement, c'est une technologie très récente, complexe et évoluant rapidement, de laquelle une effervescence a été créée depuis quelques années mais qui souffre encore de problèmes de maturité. Bien que l'implémentation de Bitcoin soit stabilisée, ce n'est pas le cas pour les technologies émergentes comme Ethereum.

Le travail à effectuer pour permettre à la technologie de se stabiliser et de se populariser reste colossale :

- le développement d'outils à destination du grand public permettant d'interagir facilement avec la blockchain ;
- l'amélioration des protocoles de consensus (preuve de travail, preuve d'enjeu, etc.) ;
- l'augmentation du nombre de transactions traitées par seconde ;
- la création et l'amélioration d'outils pour le développement d'applications basées sur l'utilisation de la blockchain ;
- la recherche sur les bonnes pratiques pour la sécurité des contrats intelligents ;
- la mise en place de protocoles de désidentification pour les transactions.

Cette liste n'est pas exhaustive et les tâches à accomplir sont énormes. Malgré tout, les personnes impliquées sur les différents projets de blockchain sont désireux de faire avancer cette technologie et la recherche progresse rapidement.

Comme conséquence naturelle de sa nouveauté, la technologie est également mal comprise. Il est possible d'avoir lu de nombreux articles sur le sujet sans en avoir réellement compris le fonctionnement ou l'intérêt. En effet, il existe énormément d'informations sur la blockchain et notamment sur les possibilités de son utilisation mais peu d'articles expliquent clairement et de manière détaillée son fonctionnement. La nouveauté de la technologie explique également celle de son écosystème, ce n'est qu'en 2013 que les premières plate-formes d'échange de crypto-monnaie sont apparues. À l'heure actuelle, le secteur est en plein essor, et de nombreuses start-ups ainsi que des initiatives de grandes entreprises ont été lancées. Dans cet environnement, il existe encore un élément totalement manquant qui est la gestion de l'identité. Sur une blockchain publique, tout le monde est anonyme. En effet, un utilisateur est uniquement identifié par une clé publique et non par son identité. L'anonymat est un avantage dans certains types d'utilisation, mais d'autres

1. Une attaque Sybil est une attaque informatique qui vise à renverser un système par la création de fausses identités dans un réseau pair-à-pair.

utilisations nécessitent une identité qui peut être certifiée et associée à un compte en particulier. Il est donc nécessaire que ce compte puisse être récupéré par son propriétaire légitime en cas de perte. Il s'agit d'une question complexe majeure découlant de la conception de la technologie. Cependant, de nombreuses initiatives sont en cours dans ce domaine comme par exemple uPort [2] de ConsenSys [3]. Depuis la création d'Ethereum, cet écosystème comprend également des entités émergentes sur la blockchain, à savoir les applications décentralisées, plus communément appelées dApp [4]. La sécurité des smart contracts est encore à un stade précoce, et des efforts considérables seront nécessaires avant d'envisager l'automatisation des opérations à très forte valeur ajoutée. De nombreuses initiatives ont déjà été lancées à cet égard et se poursuivront comme la recherche fondamentale, les logiciels d'analyse formelle, l'amélioration du langage ou encore de la machine virtuelle Ethereum. De nombreux projets dApp ont été financés cette année, en particulier par le biais de l'ICO [5].

De nombreux types d'utilisation de la blockchain sont donc en cours de développement. Ces utilisations ont besoin d'avoir des règles détaillées et un cadre juridique défini. Malheureusement, dans l'état actuel des choses, la loi n'encadre pas la technologie aussi bien au niveau de la reconnaissance juridique des nouveautés apportées qu'au niveau des restrictions ou des contrôles de son utilisation. La reconnaissance de ces caractéristiques par le droit local, européen et les traités internationaux serait une avancée majeure dans de nombreux secteurs comme la finance, la santé ou l'énergie, et permettrait aux entreprises de créer des systèmes innovants. Ce manque de reconnaissance officielle a également une incidence sur la participation du secteur privé. Si l'Europe veut être un précurseur de la révolution blockchain, elle doit s'impliquer davantage, tant au niveau national qu'international.

Dans le futur, il sera important de s'intéresser aux point suivants :

- les développements techniques majeurs tels que la vérification formelle des contrats intelligents, la preuve d'enjeu ou la gestion de l'identité ;
- la reconnaissance par le secteur public et privé des crypto-monnaies et plus généralement des enregistrements dans la blockchain ;
- les projets en cours de développement qui seront lancés à court et à moyen terme comme les projets impliquant des blockchains et les dApps.

4.2 La décentralisation du Web

Le but initial du web et de l'internet était de créer un réseau neutre commun auquel chacun puisse participer de manière égale afin d'améliorer l'humanité. Heureusement, il existe un mouvement émergent pour ramener le web à cette vision et implique même certains des éléments clés de la naissance du web. C'est ce qu'on appelle le Web décentralisé ou le Web 3.0. Il décrit une tendance émergente pour la création de services sur Internet qui ne dépendent pas d'une seule organisation centrale. La participation égale sur Internet s'est rapidement effondrée lorsque des personnes se sont rendues compte qu'un moyen simple de créer de la valeur sur ce réseau était de construire des services centralisés qui rassemblent, piègent et monétisent des informations. Les moteurs de

2. Voir https://www.uport.me/, uPort est un système d'identité entièrement autonome basé sur Ethereum.

3. Voir https://consensys.net/, ConsenSys est une entreprise développant des applications décentralisées et des outils pour le développement d'applications basées sur l'utilisation de la blockchain.

4. acronyme anglais de decentralized application

5. ICO (acronyme anglais de Initial Coin Offering) est le processus par lequel une entité innovante offre aux investisseurs certaines quantités d'une nouvelle crypto-monnaie ou crypto-token en échange d'une quantité de crypto-monnaie existante comme Bitcoin ou Ethereum.

recherche comme Google, les réseaux sociaux comme Facebook, les applications de discussion comme WhatsApp se sont imposés en fournissant des services centralisés sur Internet.

Dans le même temps, ces organisations réduisent les libertés fondamentales de l'Internet telles que la possibilité de lier du contenu via une URL ou la possibilité pour les moteurs de recherche d'indexer son contenu. Le Web décentralisé envisage un monde futur où des services tels que la communication, la monnaie, l'édition, les réseaux sociaux ou la recherche sont fournis non pas par des services centralisés appartenant à des organisations individuelles, mais par des technologies alimentées par la communauté des utilisateurs. L'idée principale de la décentralisation est que l'exploitation d'un service n'est pas confiée à une seule entreprise omnipotente. Au lieu de cela, la responsabilité du service est partagée soit par une exécution sur plusieurs serveurs fédérés, ou par l'exécution d'applications côté client s'appuyant sur un modèle distribué. Les règles qui décrivent le comportement d'un service décentralisé sont conçues pour obliger les participants à agir de manière équitable, grâce à l'utilisation de techniques cryptographiques telles que les arbres de Merkle [6] et la signature électronique pour favoriser la responsabilité des participants.

Il existe trois concepts fondamentaux que le Web décentralisé prône :

- la confidentialité, en effet la décentralisation impose une attention accrue à la confidentialité des données. Les données sont réparties sur le réseau et les technologies de chiffrement sont essentielles pour garantir que seuls les utilisateurs autorisés puissent modifier les données qui leurs appartiennent. L'accès aux données est entièrement contrôlé par le réseau grâce à des algorithmes. Ce modèle montre une césure avec les réseaux centralisés où le propriétaire du réseau a un accès complet aux données, et donc facilite l'établissement du profil du client et le ciblage publicitaire ;
- la portabilité des données, dans un environnement décentralisé, les utilisateurs possèdent leurs données et choisissent avec qui ils partagent ces données. En outre, ils en conservent le contrôle et ils peuvent les transférer d'un service à un autre librement.
- la sécurité, les environnements décentralisés sont plus sûrs contre le piratage, l'infiltration, l'acquisition et les pannes que les environnements centralisés, car ils ont été construits pour être exposés publiquement dès le départ.

De nombreux services auparavant centralisés et aujourd'hui remplacés par des services décentralisés ont montré que cette approche peut être mise en œuvre et qu'une alternative du web existant entièrement décentralisée est réalisable. Parmi les systèmes décentralisés qui ont connu un grand succès on peut par exemple citer Git, un système de gestion de versions entièrement décentralisé, remplaçant presque totalement des systèmes centralisés tels que Subversion. Au même titre, Bitcoin a démontré qu'une monnaie peut exister sans autorité centrale, en contraste avec un service centralisé tel que Paypal. Diaspora vise quant à lui à fournir une alternative décentralisée à Facebook. Cependant, la plupart de ces technologies ont toujours été laissées de côté par le grand public. Mais aujourd'hui, les utilisateurs se rendent compte que la dépendance aux plates-formes communautaires massives qui contrôlent Internet n'est pas dans leurs intérêts.

Il existe déjà une nouvelle génération de systèmes décentralisés qui ont attiré l'attention de l'industrie traditionnelle. Blockstack [7] et Ethereum montrent que la blockchain peut être plus qu'une simple crypto-monnaie. IPFS et le projet Dat [8] fournissent également des gestionnaires de données entièrement décentralisés, où la propriété et la responsabilité des données sont partagées

6. Un arbre de Merkle, aussi appelé arbre de hachage, est une structure de données contenant l'empreinte d'un volume de données.

7. Voir https://blockstack.org/, Blockstack est la première implémentation de système de noms de domaine décentralisé s'exécutant sur la blockchain Bitcoin.

8. Voir https://datproject.org/, Dat est un outil de partage de données distribuées.

par tous ceux qui y accèdent, ne reposant plus sur un hébergement centralisé.

Bien que le Web décentralisé attire l'intérêt et la passion de la communauté des développeurs, il est actuellement impossible de dire quelles nouvelles économies émergeront et quelles sortes de nouvelles technologies et services vont être inventés. La seule certitude que nous ayons est que la décentralisation du Web est une réalité et que les personnes actives autour de cette révolution soutiendront les intérêts de leurs utilisateurs.

4.3 Solutions existantes

La couche de persistance des données de la gTSL repose sur la blockchain et des concepts de décentralisation. À ce titre, l'objectif est de conserver toutes les informations relatives aux TSPs et aux TSs dans un registre sécurisé et de répliquer toutes les données de la gTSL à travers un réseau pair-à-pair [9] décentralisé. Cela permet de garantir l'intégrité et la disponibilité des informations, puisque chaque donnée est signée et est accompagnée de toutes les données qui la précédent dans le registre. La distribution de l'information à travers un réseau pair-à-pair fournit une résilience forte contre les attaques par déni de service.

Afin d'atteindre cet objectif qui permettra de pallier aux problèmes de l'architecture actuelle, plusieurs options ont été envisagées :

- stocker les données dans une blockchain publique existante ;
- stocker les données dans une blockchain privée ;
- stocker les données dans une blockchain privée, et utiliser une blockchain publique pour s'assurer de l'intégrité des données (par exemple en conservant le hash [10] de la transaction réalisée sur la blockchain privée dans une blockchain publique) ;
- stocker les données dans un système décentralisé et distribué, comme par exemple une base de données ou un système de fichiers, et utiliser une blockchain publique pour s'assurer de l'intégrité des données (par exemple en conservant le hashes [11] ou les références des données).

Cette section présente les technologies qui ont été envisagées en tant que solutions de stockage de données dans le cadre de l'implémentation de la gTSL. On y retrouve des technologies de blockchain, mais aussi des systèmes décentralisés et distribués.

4.3.1 Ripple

Ripple [12] est une crypto-monnaie s'appuyant sur un registre distribué basé sur une blockchain qui n'utilise pas de système de preuve de travail pour l'ajout de blocs. Au lieu de cela, il repose sur un mécanisme de consensus (The Ripple Protocol Consensus Algorithm, 2014) appliqué à un sous-réseau de nœuds connus et fiables. Cette technologie est surtout orientée vers les paiements, les transactions et les échanges de crypto-monnaie mais ne propose pas de réelle solution pour le stockage des données.

9. Le pair-à-pair est un modèle de réseau informatique similaire au modèle client-serveur mais où chaque client est aussi un serveur.
10. Un hash est le résultat d'une fonction de hachage qui permet d'identifier rapidement une donnée.
11. Le terme hashes est le pluriel du mot hash.
12. Voir https://ripple.com/

4.3.2 Tendermint

Tendermint [13] est une crypto-monnaie s'appuyant sur un registre distribué basé sur une blockchain qui utilise un système de votes par un consensus au lieu du minage. À ce titre, Il repose sur un ensemble de nœuds de validation dont la responsabilité est d'émettre des votes signés pour ou contre l'ajout de nouveaux blocs dans la chaîne. Le scrutin est validé si au minimum les deux tiers des nœuds de validation votent pour l'acceptation d'un nouveau bloc. Tendermint apporte uniquement une solution concernant l'utilisation d'un système de votes à la place d'une preuve de travail.

4.3.3 Ethereum

Ethereum [14] est une plate-forme publique décentralisée et distribuée basée sur la technologie blockchain. Elle repose sur des programmes dont le code et les données sont stockés sur la blockchain. Ces programmes s'appellent des smart contracts. Ethereum apporte donc l'utilisation de concepts de blockchain au-delà du cas d'utilisation de la crypto-monnaie et offre un environnement d'exécution virtuel qui peut être utilisé pour créer des organisations autonomes décentralisées, i.e. des organisations qui sont gérées par des règles spécifiées dans des smart contracts. Ethereum étant conçu principalement comme un environnement d'exécution décentralisé et autonome, il n'offre pas de fonctionnalités de stockage réelles. En effet, bien que les données puissent être stockées dans le cadre de l'exécution des contrats intelligents, le coût de son utilisation peut rapidement devenir prohibitif. En effet, le système est conçu pour calculer le coût des transactions en fonction des ressources nécessaires en termes de calcul, de bande passante et de stockage.

4.3.4 Swarm

Swarm [15] est une plate-forme de stockage distribuée ainsi qu'un service de distribution de contenu directement lié à Ethereum. Son objectif initial est de servir de solution de stockage décentralisé et redondant pour les enregistrements dans le registre publique d'Ethereum, mais il peut également être utilisé comme solution de stockage à part entière et de service pair-à-pair. Au moment de la rédaction de ce mémoire, Swarm était encore à ses débuts de développement, avec uniquement une version "alpha" disponible.

4.3.5 Hyperledger Fabric

Hyperledger Fabric [16] est une plate-forme reposant sur un registre distribué destinée à l'exécution de smart contracts. Il est conçu selon une architecture modulaire, prend en charge les contrats intelligents écrits dans le langage de programmation Go et repose sur un réseau de pairs de validation (c'est-à-dire les nœuds responsables du maintien du registre) et de pairs de non validation. À l'instar d'Ethereum, Hyperledger ne gère pas le stockage de données nativement et simplement, mais son architecture modulaire pourrait le permettre.

13. Voir https://tendermint.com/
14. Voir https://ethereum.org/
15. Voir http://swarm-gateways.net/bzz:/swarm-gateways.eth/
16. Voir https://www.hyperledger.org/projects/fabric

4.3.6 Keyless ledger

Keyless Signature Infrastructure [17] (KSI) est une plate-forme offrant une authentification basée sur la signature électronique pour les données numériques, les machines et les humains. KSI repose uniquement sur les fonctions de hachage, son registre agit comme un enregistrement de timestamps [18] émis pour les hashes de données soumises par les utilisateurs. Son intérêt est de fournir des signatures électroniques, et selon les dires des développeurs d'une manière plus sécurisée que la Public Key Infrastructure [19] (PKI).

4.3.7 OpenChain

OpenChain [20] est un registre distribué open-source qui repose uniquement sur une signature électronique qui est générée pour les transactions qu'il enregistre. Les transactions sont directement liées les unes aux autres, sans l'utilisation de blocs, et peuvent maintenir des données. OpenChain peut posséder plusieurs instances, chacune répliquant les autres. Cependant, la technologie est construite autour d'une hiérarchie de nœuds de validation et de nœuds d'observation, dans laquelle les nœuds de validation peuvent ajouter et valider des transactions pour le registre et les nœuds d'observation peuvent uniquement répliquer les données des nœuds de validation auxquelles ils sont connectés. Par conséquent, il n'est pas possible d'implémenter un réseau purement décentralisé d'instances OpenChain.

4.3.8 BigchainDB

BigchainDB [21] vise à interfacer la technologie blockchain et une base de données en ajoutant des caractéristiques de la blockchain comme la décentralisation, l'immutabilité et l'échange d'actifs à une implémentation d'une base de données NoSQL [22] existante. Cette technologie en est encore à ses débuts de développement et manque actuellement de contrôles de sécurité de base (par exemple, un administrateur de base de données supprimant une base de données sur un noeud verra cette opération être répliquée sur tous les autres noeuds). De plus, BigChainDB n'est pas Byzantine Fault Tolerant [23] (BFT).

4.3.9 InterPlanetary File System (IPFS)

IPFS [24] est un système de fichiers distribué pair-à-pair qui cherche à connecter tous les périphériques informatiques avec un même système de fichiers. Il réutilise le paradigme de la blockchain,

17. Voir https://guardtime.com/technology/ksi-technology
18. Le timestamping, aussi nommé horodatage en français, est un mécanisme qui consiste à associer une date et une heure à un événement, une information ou une donnée informatique.
19. La PKI est une infrastructure visant à fournir une garantie de confiance dans la validité d'une identité numérique et utilisant pour cela une pair de clé liée à un certificat.
20. Voir https://www.openchain.org/
21. Voir https://www.bigchaindb.com/
22. NoSQL, pour Not only SQL, est une famille de systèmes de gestion de base de données (SGBD) qui s'écarte du paradigme classique des bases relationnelles.
23. La Byzantine fault tolerance (BFT) est la caractéristique d'un système qui tolère la classe de défaillances connue sous le nom du problème des généraux byzantins pour lesquels il existe une preuve de non-solvabilité
24. Voir https://ipfs.io/

plus précisément les concepts d'immutabilité des données et de la décentralisation réalisée à travers une communication pair-à-pair, tout en se basant sur le système de contrôle de version Git [25]. Il permet notamment de stocker des informations, de tous types et de tous volumes, qui sont répliquées à travers le réseau.

4.3.10 Monax

Monax [26] est une plate-forme open-source qui vise les développeurs qui veulent construire et exécuter des applications basées sur la blockchain pour des écosystèmes business. Il peut être comparé à Ethereum, toutefois avec des autorisations qui le rendent juridiquement utilisable dans les environnements commerciaux. Cette technologie a pour intérêt de mettre en place sa propre plate-forme de blockchain dans un environnement business.

4.3.11 Factom

Factom [27] fournit un protocole distribué et décentralisé qui s'exécute sur la blockchain Bitcoin, et qui maintient un registre inaltérable. Cette technologie a pour but de conserver les documents des entreprises de manière sécurisée.

4.3.12 Emercoin

Emercoin [28] est une crypto-monnaie utilisant un minage qui repose sur la preuve de travail et la preuve par votes [29]. Il diverge des crypto-monnaies "standard" dans le sens où sa blockchain ne se limite pas à une utilisation de registre de transactions. Hormis l'utilisation de la crypto-monnaie, d'autres services sont supportés comme un système de noms de domaine décentralisé ou un stockage sécurisé pour des timestamps. Malgré tout, il n'est pas conçu pour le stockage de données.

4.4 Synthèse

Nous avons effectué les choix technologiques inhérents à l'architecture de notre système après une analyse de la technologie blockchain, et suite à une prise en considération de ses avantages et inconvénients.

4.4.1 Avantages de la blockchain

Si cette technologie connaît un tel succès c'est parce qu'elle apporte de nombreux avantages :

25. Voir https://ripple.com/
26. Voir https://monax.io/
27. Voir https://www.factom.com/
28. Voir https://emercoin.com/
29. en anglais, Proof of Stake.

— la décentralisation, qui signifie que son architecture ne repose pas sur une entité centrale et permet d'enregistrer des données dans un réseau distribué ;
— la transparence, puisque l'état des données conservées est consultable publiquement par tout le monde ;
— l'autonomie, puisqu'elle est basée sur un consensus dans lequel chaque partie prenante peut transférer des données de manière sécurisée et autonome ;
— l'immutabilité, en effet toute transaction est persistée définitivement et donc ne peut être effacée ;
— l'anonymat, car tout utilisateur de la blockchain est anonyme dans le sens où il n'est pas désignée par son identité mais uniquement par une clé publique [30].

4.4.2 Inconvénients de la blockchain

Bien que la blockchain apporte de nombreux avantages, elle comporte aussi des inconvénients :
— la performance, en effet cette technologie est et sera toujours plus lente qu'une base de données centralisée puisqu'elle nécessite pour chaque transaction une vérification de signature, une validation par le consensus et la redondance des informations ;
— la consommation énergétique, puisque la validation de blocs reposent sur la résolution d'un puzzle cryptographique nécessitant une grande puissance de calcul ;
— le coût, dans le cas où il est nécessaire d'effectuer un grand nombre de transactions coûteuses ;
— la confidentialité, puisque toute information enregistrée dans la blockchain est publique, il est fortement déconseillé d'y stocker des informations confidentielles ou personnelles, même si elles sont chiffrées.

Le problème du coût des transactions d'une blockchain dans le cadre du stockage de données

Toutes les technologies énoncées ci-dessus ont été envisagées dans le cadre du stockage des données liées à la gTSL. Toutes les données doivent être stockées dans un réseau décentralisé et distribué, et doivent être publiques. L'inconvénient majeur des technologies de blockchain est le coût, en matière de crypto-monnaie, du stockage d'un grand volume de données. En effet, comme énoncé en Section 4.3.3, la preuve de travail qui permet de sécuriser la création de nouveaux blocs dans la blockchain a été conçu pour calculer le coût des transactions en fonction des ressources nécessaires en termes de calcul, de bande passante et de stockage. Si l'on prend l'exemple de la blockchain Ethereum, en se basant sur le prix de l'ether [31] (ETH) qui est d'environ 250€ au moment de la rédaction de ce mémoire, le stockage de 1Go de données coûterait 640 000€.

4.4.3 Le choix opéré

Dans cette section sont détaillées les différentes solutions préconisées, présentées en Section 4.3, dans le cadre du stockage des données de la gTSL. Pour chacune d'entre elles sont exposés ses avantages et ses inconvénients. Le choix opéré a été validé par la réalisation d'une preuve de concept présentée en Section 6.2.1.

30. Une clé publique est un encodage rendu public dans le cadre d'un échange d'informations utilisant le principe de la cryptographie asymétrique.
31. L'ether (ETH) est la crypto-monnaie utilisée sur la blockchain Ethereum.

Stockage dans une blockchain publique

La première solution de stocker les données directement dans une blockchain publique existante a l'avantage d'être relativement simple à mettre en place puisqu'elle nécessite uniquement d'être connecté au réseau de la blockchain et de soumettre des transactions contenant les données. Malgré cela, la technologie blockchain ne vise pas à stocker des grands volumes de données, ce qui explique le coût de stockage élevé qui rend son utilisation en tant que base de données inefficace.

Stockage dans une blockchain privée

La seconde solution de stocker les données dans une blockchain privée a l'avantage d'être relativement peu coûteuse puisque la blockchain est fermée à un nombre restreint de nœuds connus qui ont à charge d'en maintenir l'état. L'inconvénient est que le consensus d'une blockchain privée est faible ce qui le rend vulnérable aux attaques Sybil. En effet, le nombre de nœuds honnêtes est relativement faible ce qui engage la sécurité et l'intégrité des données. Si l'on prend l'exemple d'une blockchain privée maintenue par dix nœuds considérés honnêtes où leur poids de vote est d'une valeur égale à 1, alors un attaquant qui arriverait à connecter onze autres nœuds à cette blockchain (en considérant que ces nœuds ont aussi un poids de vote égal à 1) deviendra majoritaire et pourra corrompre les données de la blockchain puisqu'il a brisé le consensus. Cela est beaucoup plus difficile à produire dans une blockchain publique comme Ethereum car le nombre de nœuds honnêtes est très important.

Stockage dans une blockchain privée, et utilisation d'une blockchain publique pour assurer l'intégrité des données

Une autre solution envisagée est de stocker les données dans une blockchain privée, et d'utiliser une blockchain publique pour s'assurer de l'intégrité des données, mais cela complexifie l'architecture par l'utilisation de deux blockchains. De plus, comme dit précédemment une blockchain n'est pas conçu pour être utilisé en tant que base de données, car un des inconvénients est la performance, en effet cette technologie sera toujours plus lente qu'une base de données puisqu'elle nécessite pour chaque transaction une vérification de signature et une validation par le consensus.

Stockage dans un système décentralisé et distribué, et utilisation d'une blockchain publique pour assurer l'intégrité des données

La dernière solution de stocker les données dans un système de fichiers décentralisé et distribué et d'utiliser une blockchain publique afin de s'assurer de l'intégrité des données est la solution la plus pertinente dans le cadre de la mise en place de la gTSL. IPFS a été choisi en tant que système de fichiers, cette technologie a l'avantage d'être "content-addressable", c'est-à-dire que l'on peut récupérer les données grâce à l'utilisation d'un hash. Ethereum a été choisi en tant que blockchain, elle a pour rôle de conserver les hash des données. L'avantage d'un système de fichiers décentralisé et distribué est qu'il permet de stocker des grands volumes de données qui seront automatiquement répliqués à travers les nœuds du réseau. Cela apporte une forte résilience contre les attaques par déni de service et évite un point individuel de défaillance. L'avantage de la blockchain est qu'elle permet la transparence des données dans un réseau publique, qu'elle assure

l'intégrité et l'immutabilité des données grâce à un consensus et qu'elle fournit un anonymat aux utilisateurs. Ce système permet donc de réduire les coûts puisque l'on stocke uniquement un hash servant à identifier les données dans la blockchain. Ce hash a une taille de quelques octets seulement. Malgré cela, l'architecture est complexifiée puisque l'on interface deux technologies distribuées et il est donc nécessaire de déployer à la fois un nœud IPFS et un nœud Ethereum.

5 Analyse du problème et solution élaborée

Le système de gTSL a été réalisé dans le cadre du projet FutureTrust qui a pour objectif de faciliter l'utilisation de l'identification et de la signature électronique, et qui vise plus particulièrement à mettre en application le règlement de l'UE sur l'identification électronique et les services de confiance pour les transactions électroniques sécurisées au sein de l'UE. La solution élaborée consiste à étendre l'infrastructure de la liste européenne de services de confiance existante. L'objectif est de mettre en place un système utilisant une architecture décentralisée basée sur la blockchain afin de conserver et de distribuer les informations relatives aux listes de confiance, nommées Trust Service List (TSL). Dans ce chapitre est détaillée l'analyse du système à implémenter afin de répondre aux besoins énoncés dans la Section 3.1 et de pallier aux problèmes de l'architecture actuelle énoncés dans la Section 3.2.

Dans ce chapitre, nous détaillons les technologies résultantes du choix opéré présenté en Section 4.4.3. Ensuite, nous exposons les différents acteurs du système ainsi que leurs rôles. Puis, nous présentons les cas d'utilisation de la gTSL. Enfin, les modules du système, les processus métier ainsi que le modèle de données sont développés.

5.1 Description détaillée des technologies utilisées

Cette section a pour objectif d'apporter des précisions sur l'utilisation des technologies IPFS et Ethereum afin de comprendre leur fonctionnement.

5.1.1 Ethereum

Ethereum [8] est une plate-forme de blockchain open-source qui permet à qui le souhaite de créer et d'utiliser des applications décentralisées qui s'exécutent sur la blockchain. La particularité de la blockchain Ethereum est qu'elle a été conçue pour permettre aux utilisateurs d'écrire et d'exécuter des smart contracts. Un smart contract décrit un code informatique qui permet de faciliter l'échange d'argent, de contenu, de propriété, de partage ou de valeur sur la blockchain. Lors de son exécution, le contrat devient un programme informatique autonome qui s'exécute automatiquement lorsque des conditions spécifiques sont remplies. Étant donné que les smart contracts fonctionnent sur la blockchain, ils s'exécutent sans possibilité de censure, de temps d'arrêt ou de fraude.

L'innovation fondamentale d'Ethereum réside dans sa machine virtuelle appelée Ethereum Virtual Machine (EVM) qui est un système Turing-complet fonctionnant sur le réseau Ethereum. Elle permet à quiconque d'exécuter des smart contracts, quel que soit le langage de programmation, en fournissant suffisamment de temps et de mémoire. Le langage le plus utilisé sur la blockchain

Ethereum est Solidity, qui une fois compilé peut être exécuté par l'EVM.

L'avantage majeur d'exécuter des applications sur une blockchain est de bénéficier des propriétés suivantes :

- immuable, puisque les données ne peuvent être modifiées ;
- infalsifiable, en effet les applications sont basées sur un réseau reposant sur un consensus, ce qui rend la falsification impossible ;
- sécurisée, sans point individuel de défaillance et sécurisé grâce à la cryptographie, les applications sont protégées contre les attaques par déni de service et les activités frauduleuses ;
- pleine disponibilité, puisque les applications ne s'arrêtent jamais et ne peuvent jamais être arrêtées.

5.1.2 IPFS

IPFS [4] est un système de fichiers distribué pair-à-pair qui vise à connecter tous les périphériques informatiques avec un même système de fichiers. Il expose une plate-forme pour l'écriture et le déploiement d'applications, et un nouveau système de distribution pour les grands volumes de données. En d'autres termes, IPFS a pour objectif de fournir un modèle de stockage basé sur des blocs dit "content-addressable" par l'utilisation de hashes calculés sur les données. Il est conçu comme un arbre de Merkle, une structure de données à partir de laquelle il est possible de créer des systèmes de fichiers ou même des blockchains. IPFS ne contient pas de point individuel de défaillance puisqu'il est décentralisé, aucun nœud n'est privilégié et les nœuds n'ont pas besoin de se faire confiance. Chaque nœud maintient une partie des données du réseau dans son espace de stockage local. Les nœuds sont interconnectés et échangent des objets. Ces objets représentent des fichiers et d'autres structures de données.

Le protocole IPFS est divisé en une pile de sous-protocoles responsables de différentes fonctionnalités :

- identités, gère la génération et la vérification de l'identité des nœuds ;
- réseau, gère les connexions aux autres nœuds et utilise divers protocoles de réseau sous-jacents ;
- routage, maintient des informations pour localiser des pairs et répond aux requêtes locales et distantes des utilisateurs ;
- échange, est un protocole d'échange de blocs qui contrôle la distribution efficace des blocs ;
- objets, est un arbre de Merkle d'objets immuables adressés par leur contenu (hash) ;
- fichiers, est un système de fichiers hiérarchique inspirée de Git ;
- nommage, est un système de nommage mutable auto-signé.

Ces sous-systèmes ne sont pas indépendants. Ils sont interfacés et exploitent des propriétés communes.

5.2 Acteurs

Un acteur est défini comme étant un ensemble cohérent de rôles que les utilisateurs du système peuvent avoir lorsqu'ils interagissent avec celui-ci. Un acteur peut être soit un individu, soit un système externe. Dans le contexte de la gTSL, on identifie deux acteurs potentiels.

5.2.1 External User

External User[1] représente un utilisateur sans privilèges spécifiques qui souhaitent interagir avec le système, dans le but de récupérer des informations relatives à la gTSL. Il a accès au *Global Trust Service Responder* et peut donc valider le statut qualifié d'un TS ou d'un TSP donné.

5.2.2 Administrator User

Administrator User[2] représente un utilisateur externe disposant des droits requis et nécessaires pour effectuer des opérations de gestion sur la gTSL, comme par exemple mettre à jour les informations d'un TSP. Il a accès à l'interface d'administration et à ses fonctionnalités d'édition des listes de confiance. L'utilisateur d'administration hérite de l'utilisateur externe du point de vue UML[3].

5.3 Diagramme de cas d'utilisation

La Figure 5.1 présente les cas d'utilisation de la gTSL, groupés par catégorie et associés aux acteurs réalisant les actions. À la suite de cette figure sont détaillés tous les cas d'utilisation.

1. en français, utilisateur externe.
2. en français, utilisateur d'administration.
3. UML (acronyme anglais de Unified Modeling Language) est un langage de modélisation utilisé pour la conception de systèmes d'information.

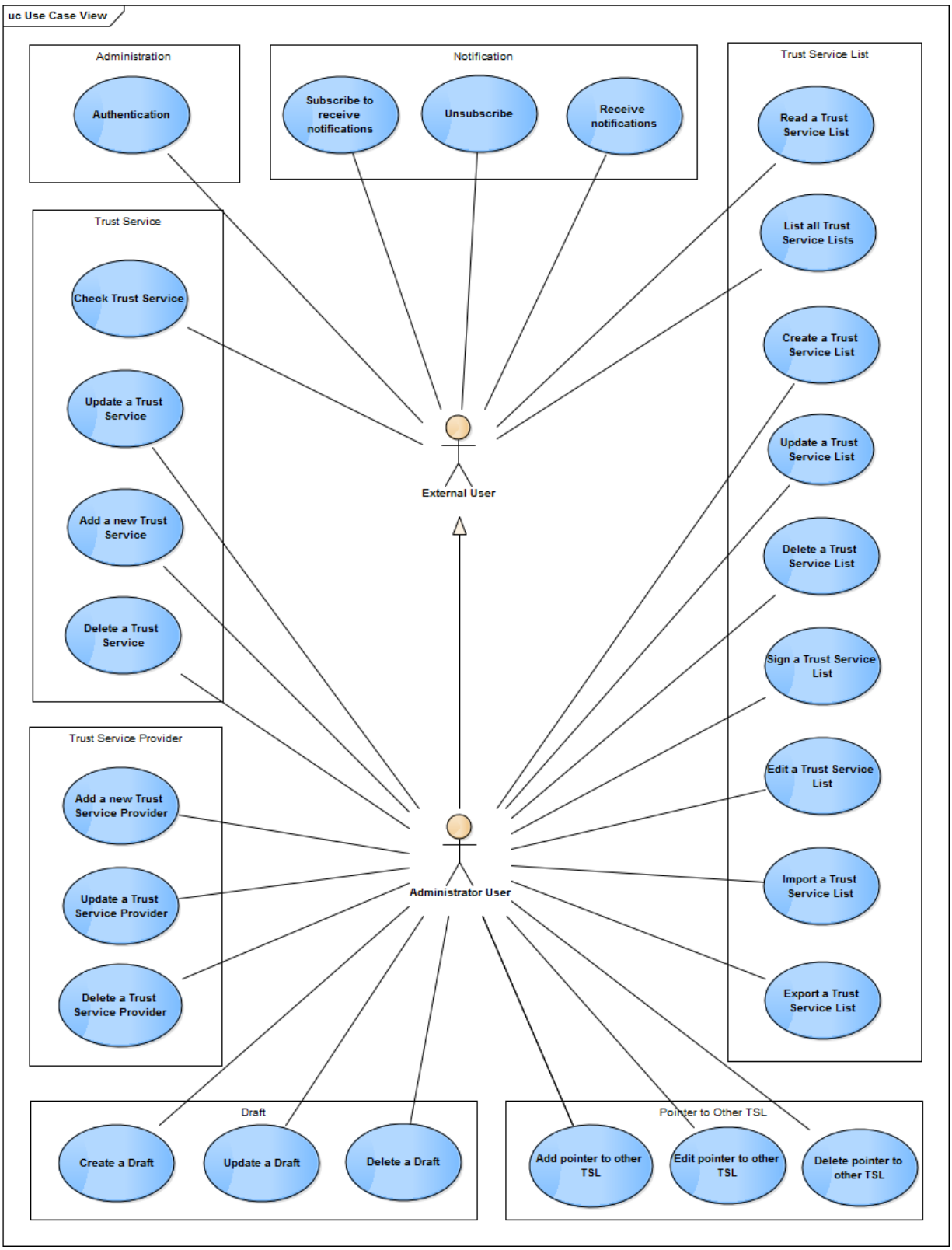


FIGURE 5.1 – Diagramme de cas d'utilisation. Illustration adaptée de *Design documentation* [7]

5.3.1 Administration

Authentification

L'authentification permet aux utilisateurs d'être reconnus par le système comme étant administrateur, c'est-à-dire comme ayant droit d'éditer des listes de confiance. Cette fonctionnalité est gérée grâce à la blockchain, en effet les clés publiques des utilisateurs enregistrés en tant qu'administrateur sont sauvegardées dans la blockchain. Afin de s'authentifier, l'utilisateur doit donc utiliser sa clé privée.

5.3.2 Trust Service List

Créer une nouvelle Trust Service List

Un utilisateur authentifié et autorisé peut créer une TSL pour un territoire donné. Pour cela, il doit créer une nouvelle TSL, remplir tous les champs requis de celle-ci, valider ces champs et enfin la signer. Ensuite, la liste complète est stockée et répliquée à travers les nœuds du réseau distribué et une nouvelle entrée est créée dans la blockchain afin de la référencer dans la gTSL.

Lire les informations relatives à une Trust Service List

Tout utilisateur peut lire et récupérer les informations relatives à une TSL existante afin d'en analyser le contenu. Le système permet d'effectuer une recherche en se basant sur différents critères comme par exemple le territoire, le type de service, le statut ou un certificat.

Éditer une Trust Service List

Un utilisateur authentifié et autorisé peut modifier une TSL. Ce processus est similaire à la création sauf que l'utilisateur n'a pas à saisir toutes les informations car la liste existante lui est fournie. Une édition peut donner lieu à l'ajout, la suppression ou la modification d'un TSP ; l'ajout, la suppression ou la modification d'un TS ; la modification d'un champ obligatoire ; l'ajout, la suppression ou la modification d'un champ optionnel. Il est important de noter que lors d'une édition, l'utilisateur doit signer à nouveau la TSL.

Supprimer une Trust Service List

Un utilisateur authentifié et autorisé peut supprimer une TSL. Ce cas d'utilisation correspond à la révocation d'une TSL. Dans la pratique, une TSL n'est jamais supprimée définitivement, elle est simplement considérée comme révoquée et peut être possiblement réhabilitée.

Lister toutes les Trust Service Lists

Le système permet la récupération de l'ensemble des TSLs afin d'avoir une vue globale de la gTSL.

Signer une Trust Service List

Cette fonctionnalité permet aux utilisateurs authentifiés et autorisés de signer une TSL lors de la création ou d'une édition.

Importer une Trust Service List

Le système permet aux utilisateurs d'importer des TSLs au format XML afin de les créer ou les modifier dans la gTSL. Cela a pour intérêt d'une part de permettre une édition à la main des utilisateurs, d'autre part d'autoriser un système externe de créer des TSLs. Malgré cela, la TSL doit être valide afin d'être acceptée par le système. Cette fonctionnalité peut aussi être utile en cas de migration. Il est important de noter que le fichier XML doit être signé.

Exporter une Trust Service List

Le système permet aux utilisateurs d'exporter des TSLs au format XML. Cela pour intérêt de permettre à des systèmes externes de les manipuler et d'y appliquer des modifications. Cette fonctionnalité peut aussi être utile en cas de migration.

5.3.3 Draft

Créer un brouillon de Trust Service List

Le système permet aux utilisateurs de créer un brouillon [4] d'une TSL. En effet, un utilisateur peut créer un brouillon d'une liste existante dans le but de la soumettre plus tard. Les brouillons sont stockés dans une base de données locale. Lorsque que l'utilisateur considère son brouillon comme terminé et que le système l'a validé, le brouillon est alors poussé en production et remplace la liste actuelle. On peut assimiler ce fonctionnement à celui des emails, qui peuvent être enregistrés comme brouillon avant d'être envoyés.

Éditer un brouillon de Trust Service List

Le système permet aux utilisateurs d'éditer un brouillon qui a été enregistré localement mais qui n'a pas été soumis à la production.

Supprimer un brouillon de Trust Service List

Le système permet aux utilisateurs de supprimer un brouillon qui a été enregistré localement mais qui n'a pas été soumis à la production.

4. en anglais, draft.

5.3.4 Pointer to Other TSL

Ajouter un pointeur vers une autre TSL

Dans l'architecture actuelle, il est possible d'ajouter à une TSL un pointeur vers une autre TSL. Un pointeur permet de référencer une liste dans une autre lorsque cela est pertinent. Afin de rester conforme au format actuel de Trust Service List, cette action doit être disponible dans la nouvelle implémentation.

Éditer un pointeur vers une autre TSL

Un utilisateur authentifié et autorisé peut modifier un pointeur d'une TSL. En effet, si une TSL est modifiée, il est possible que sa référence soit aussi modifiée.

Supprimer un pointeur vers une autre TSL

Un utilisateur authentifié et autorisé peut supprimer un pointeur d'une TSL. Dans le cas où il n'est plus pertinent de référencer une autre TSL, l'utilisateur peut supprimer le pointeur.

5.3.5 Trust Service Provider

Ajouter un nouveau Trust Service Provider

Un utilisateur authentifié et autorisé peut ajouter un TSP dans une TSL. Pour cela, il doit créer un nouveau fournisseur, remplir tous les champs requis pour celui-ci et valider ces champs. Le fournisseur doit obligatoirement proposer au minimum un service de confiance, car dans le cas contraire il ne serait pas pertinent de l'ajouter. La liste sera alors mise à jour. Il est important de noter que l'ajout d'un fournisseur est considéré comme une édition de la liste. Par conséquent, l'utilisateur doit signer à nouveau la TSL.

Éditer un Trust Service Provider

Un utilisateur authentifié et autorisé peut éditer un TSP dans une TSL. En effet, les informations d'un TSP peuvent être amené à changer, l'utilisateur a donc la possibilité de les modifier. Une édition peut donner lieu à la modification du nom, de l'adresse électronique ou postale, de l'URI ou d'autres informations optionnelles du TSP. Tout comme l'ajout, l'édition d'un TSP est considérée comme une édition de la liste. Par conséquent, l'utilisateur doit signer à nouveau la TSL.

Supprimer un Trust Service Provider

Un utilisateur authentifié et autorisé peut supprimer un TSP d'une TSL. Par exemple, un TSP peut être supprimé s'il n'existe plus ou s'il ne répond plus aux critères de confiance. Tout comme l'ajout, la suppression d'un TSP est considérée comme une édition de la liste. Par conséquent, l'utilisateur doit signer à nouveau la TSL.

5.3.6 Trust Service

Ajouter un nouveau Trust Service

Un utilisateur authentifié et autorisé peut ajouter un TS à un TSP. Il est obligatoire que le TSP ait déjà été créé. Pour cela, il doit créer un nouveau service, remplir tous les champs requis pour celui-ci et valider ces champs. Il est important de noter que l'ajout d'un service est considéré comme une édition de la liste. Par conséquent, l'utilisateur doit signer à nouveau la TSL.

Éditer un Trust Service

Un utilisateur authentifié et autorisé peut éditer un TS d'un TSP. En effet, les informations d'un TS peuvent être amené à changer, l'utilisateur a donc la possibilité de les modifier. Une édition peut donner lieu à la modification du nom, du statut, de l'URI ou d'autres informations du TS. Tout comme l'ajout, l'édition d'un TS est considérée comme une édition de la liste. Par conséquent, l'utilisateur doit signer à nouveau la TSL.

Supprimer un Trust Service

Un utilisateur authentifié et autorisé peut supprimer un TS d'un TSP. Par exemple, un TS peut être supprimé s'il n'est plus proposé par le fournisseur. Tout comme l'ajout, la suppression d'un TS est considérée comme une édition de la liste. Par conséquent, l'utilisateur doit signer à nouveau la TSL.

Vérifier un Trust Service

Le système permet aux utilisateurs externes de rechercher une trust anchor [5] à un moment donné. Une opération de recherche peut être effectuée en passant en paramètre un certificat ou le nom associé au service souhaité d'une TSL. La réponse contiendra les informations sur le service identifié.

5.3.7 Notification

S'abonner pour recevoir des notifications

Le système permet aux utilisateurs de s'abonner à des notifications concernant les listes de confiance. Par exemple, un utilisateur peut être notifié par email en cas de modification d'une liste donnée.

Se désabonner

Un utilisateur abonné à des notifications à la possibilité de se désabonner.

5. Dans les systèmes cryptographiques à structure hiérarchique, une trust anchor est une autorité pour laquelle la confiance est assumée et non dérivée. Dans le cadre de l'architecture X.509, un certificat racine serait la trust anchor de laquelle toute la chaîne de confiance est dérivée.

Recevoir les notifications

Le système doit envoyer des notifications à tous les utilisateurs abonnés à une liste donnée, lors de la mise à jour de celle-ci.

5.4 Modules du système

Cette section décrit la structure et les composants de haut niveau. La Figure 5.2 présente les différents modules.

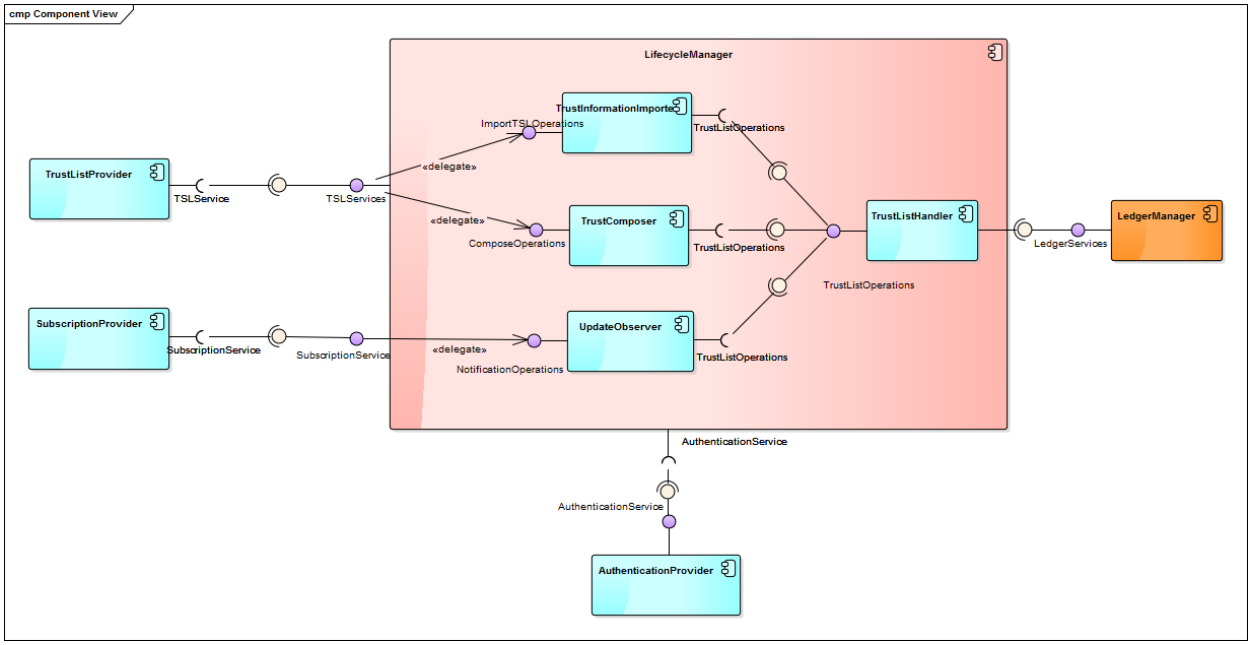


FIGURE 5.2 – Composants de haut niveau de la gTSL. Illustration extraite de *Design documentation* [7]

5.4.1 Lifecycle Manager

Ce composant est le cœur de la gTSL. Il fournit les fonctionnalités nécessaires à la gestion des listes de confiance et de leurs informations. Il est composé de quatre sous-composants :

- *TrustListHandler*, qui communique avec le *Ledger Manager*, et expose les fonctionnalités requises à la création, l'édition, la suppression et la récupération des informations concernant les TSPs et les TSs conservées dans la gTSL ;
- *TrustInformationImporter*, qui fournit les fonctionnalités requises à l'import et à l'export de listes de confiance ;
- *UpdateObserver*, implémentant un design pattern Observer [6], qui a pour rôle de notifier les utilisateurs abonnés lorsqu'un changement se produit sur la gTSL ;
- *TrustComposer*, qui gère la logique métier associée à la gestion des services de confiance et des certificats qualifiés.

6. Le design pattern (que l'on peut traduire par patron de conception) observateur est utilisé pour envoyer un signal à tous les observateurs lorsqu'une condition est remplie sur le module dit observé, la condition peut être par exemple la mise à jour d'une donnée.

De plus, le *Lifecycle Manager* est interfacé avec des composants externes :

- *TrustListProvider*, qui gérer les opérations relatives aux listes de confiance et qui est interfacé avec le *Lifecycle Manager* à travers le *TSLService*
- *SubscriptionProvider*, qui est un composant externe interfacé avec le module *UpdateObserver* à travers le *SubscriptionService*, qui fournit aux abonnés des fonctionnalités de notifications ;
- *AuthenticationProvider*, qui traite les opérations d'authentification et qui est interfacé avec le *Lifecycle Manager* à travers le *AuthenticationService*.

5.4.2 Ledger Manager

Le *Ledger Manager* est le module qui gère toutes les interactions avec la solution de stockage sur laquelle repose la gTSL pour conserver les données. Ce module s'appuie sur une interface définissant les fonctions permettant de manipuler la solution de stockage de données.

5.5 Processus métier

Cette section contient des diagrammes permettant de décrire le comportement et les interactions des différents composants, modules et classes du logiciel. Les processus présentés sont les plus représentatifs des actions possibles sur la gTSL, mais ne représentent pas la totalité des processus.

5.5.1 Édition d'une liste de confiance

À travers le processus d'édition, un utilisateur d'administration peut effectuer des modifications sur une TSL spécifiée. Ce processus est illustré dans le diagramme BPMN1.1 [7] dans la Figure 5.3.

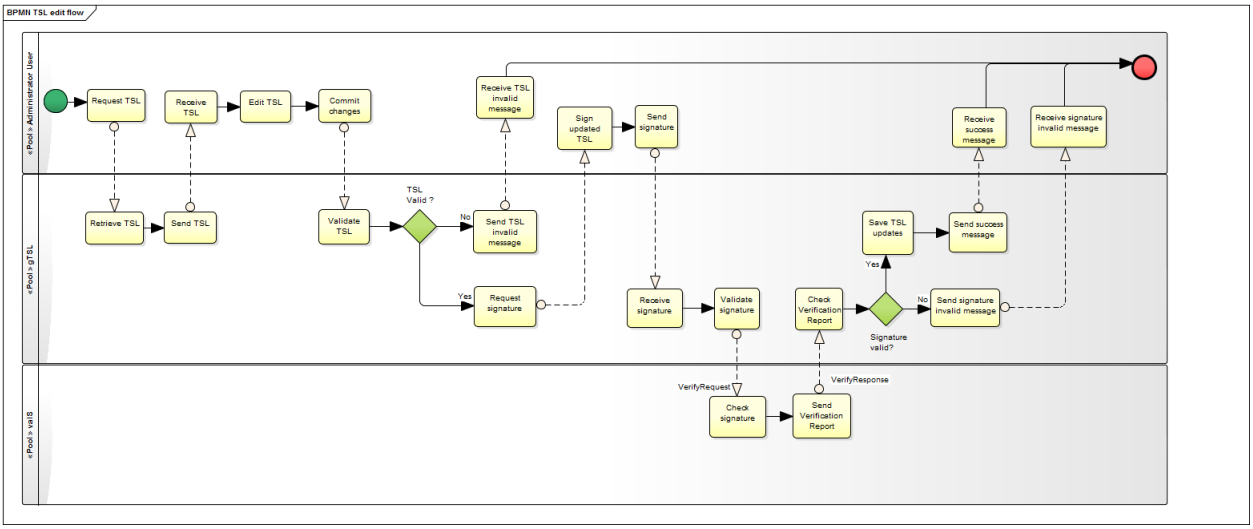


FIGURE 5.3 – Édition d'une liste de confiance. Illustration extraite de *Design documentation* [7]

7. BPMN, acronyme anglais de Business process model and notation, est une représentation graphique pour spécifier les processus métier dans un modèle de processus métier.

5.5.2 Import d'une liste de de confiance

À travers le processus d'import, un utilisateur d'administration peut importer une TSL dans la gTSL. À ce titre, un utilisateur d'administration peut soit importer une TSL existante, c'est-à-dire en téléchargeant un fichier de TSL, soit définir l'URL d'une TSL distante à inclure dans la gTSL. Ce processus est illustré dans le diagramme BPMN1.1 dans la Figure 5.4.

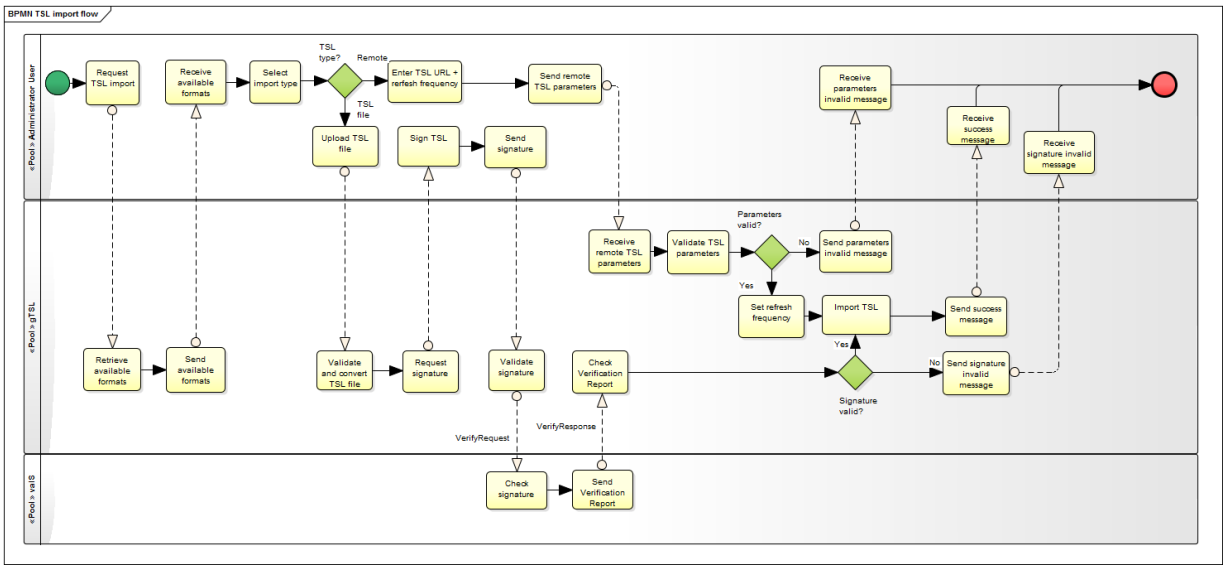


FIGURE 5.4 – Import d'une liste de confiance. Illustration extraite de *Design documentation* [7]

5.5.3 Gestion des notifications

À travers le processus de notifications, tous les utilisateurs peuvent souscrire aux notifications d'une TSL. Le système de notifications surveille les changements opérés sur la gTSL et informe les utilisateurs abonnés lorsque des modifications sont détectées. Ce processus est illustré dans le diagramme BPMN1.1 dans la Figure 5.5.

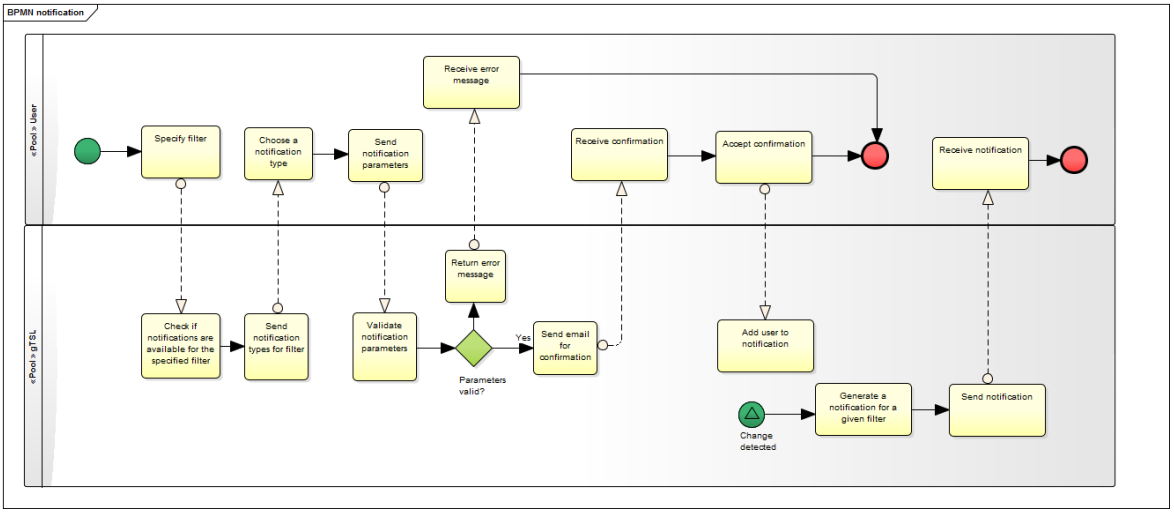


FIGURE 5.5 – Gestion des notifications. Illustration extraite de *Design documentation* [7]

5.6 Modèle des données

Cette section contient des diagrammes permettant de décrire les classes implémentées dans le logiciel. Le diagramme de classes UML fourni en Figure 5.6 définit les structures de données de base pour les listes de confiance conformément à leur définition dans ETSI TS 119 612 [11].

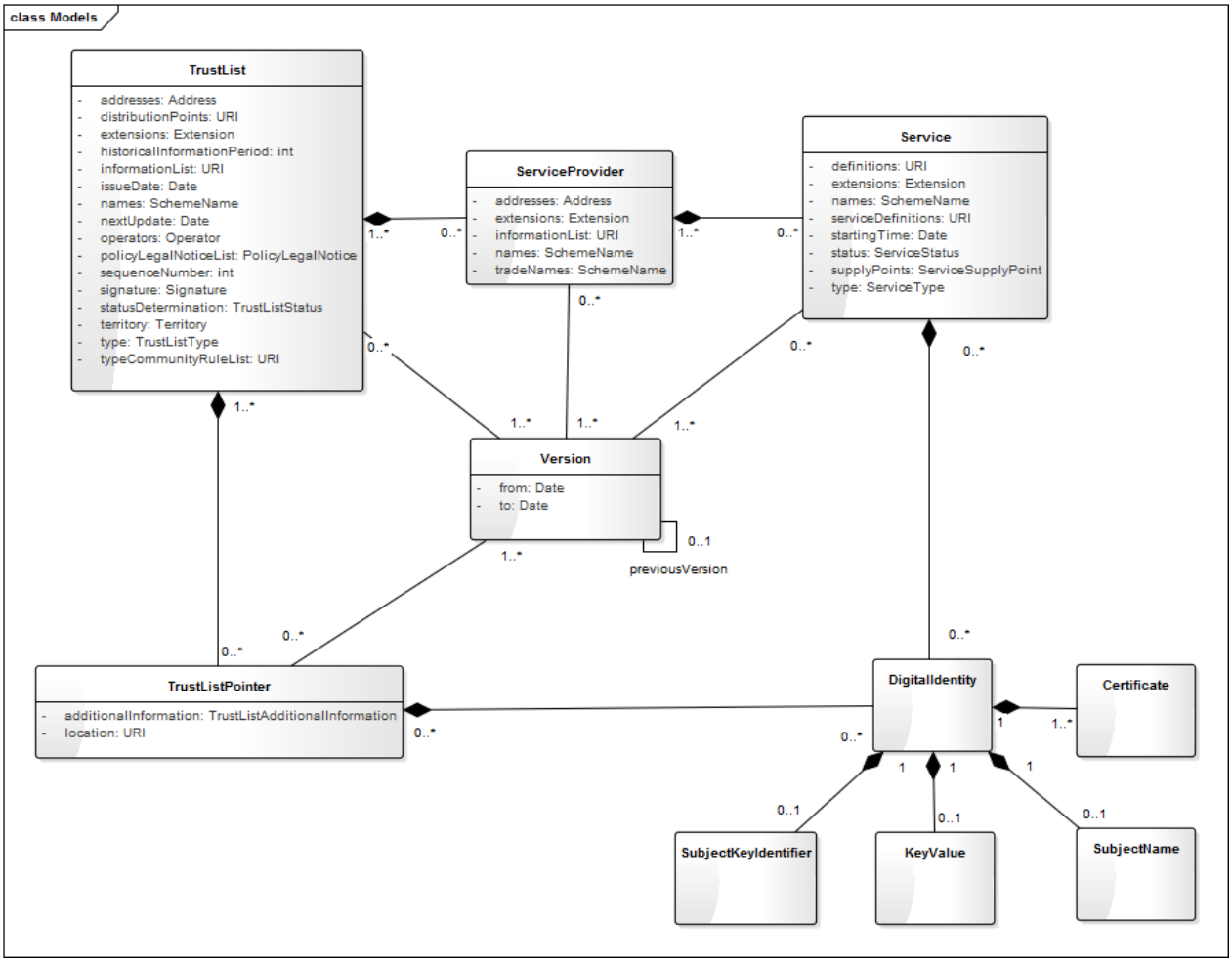


FIGURE 5.6 – Diagramme de classes simplifié. Illustration extraite de *Design documentation* [7]

Les principales entités sont :

- *TrustList*, qui définit la liste de confiance pour un territoire donné ;
- *TrustListPointer*, qui est lié uniquement aux listes de confiance des États membres et utilisé uniquement pour la LoTL ;
- *ServiceProvider*, qui contient les informations d'un fournisseur de services ;
- *Service*, qui définit un service métier pour un fournisseur identifié par un ensemble d'identités numériques ;
- *Version*, qui rassemble les informations concernant l'historique de la liste de confiance et des entités associées.

Le modèle de données peut être simplifié par un graphe orienté, comme le montre la Figure 5.7, où les nœuds représentent les entités et les arrêtes représentent les relations entre ces entités.

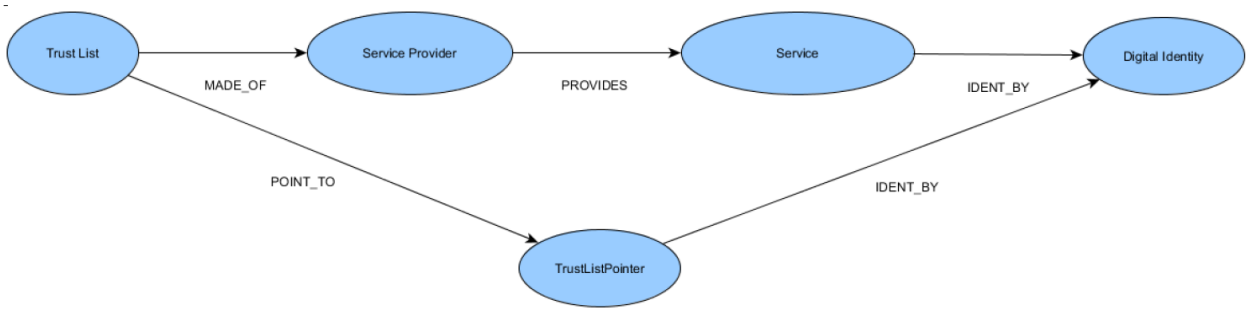


FIGURE 5.7 – Graphe de dépendances des données. Illustration extraite de *Design documentation* [7]

6 Réalisation de la solution proposée

Dans ce chapitre, nous détaillons l'architecture mise en place ainsi que l'implémentation des modules présentés au Chapitre 5. Ce chapitre présente aussi la solution et les validations qui ont été effectuées sur celle-ci.

6.1 Architecture mise en place

Cette partie concerne l'architecture du système et différencie d'une part l'architecture d'un nœud de la gTSL et d'autre part l'architecture globale du système. La première vision permet de détailler au niveau local les différents composants utilisés. La seconde vision permet de comprendre au niveau général la décentralisation et la distribution des informations à travers un réseau de multiples nœuds interconnectés de la gTSL, agissant dans un but commun.

Cette section présente uniquement l'architecture logicielle du service de gTSL mais ne fournit pas la documentation de la mise en place technique du système. Des explications concernant le déploiement du système (outils utilisés, processus d'installation, etc.) sont détaillées dans le document GTSL - Project Setup Documentation disponible en Annexe C. J'ai eu pour mission de rédiger ce document dans le but de fournir une précision sur le déploiement du module de gTSL. Néanmoins, ce document ne correspond pas au manuel d'installation utilisateur qui fait l'objet d'un second document intitulé GTSL - Project Setup Guidelines disponible en Annexe D que j'ai également rédigé.

6.1.1 Architecture d'un nœud de la gTSL

Le système mis en place est composé de deux parties principales. La première partie est une interface d'administration qui permet aux utilisateurs authentifiés d'opérer facilement des modifications sur la gTSL à l'aide d'une interface graphique. La seconde partie est un nœud de la gTSL qui contient un serveur d'applications, une base de données locale, une instance connectée au réseau IPFS et une instance connectée à la blockchain Ethereum. Ces deux parties communiquent via le protocole HTTP. Les données que contiennent les requêtes échangées sont formatées en JSON. Il est important de noter que l'interface d'administration peut être omise dans le cas où le nœud est déployé à des fins de consultation de la gTSL par des utilisateurs externes uniquement. La Figure 6.1 présente les interactions entre ces différents composants.

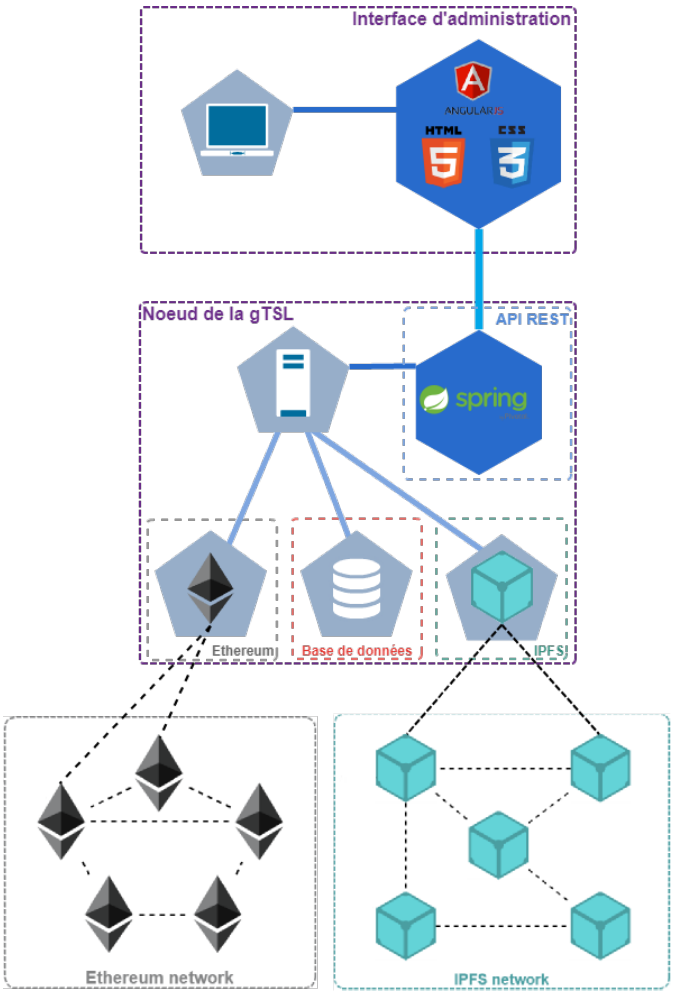


FIGURE 6.1 – Architecture du système

Interface d'administration

L'interface d'administration fournit une interface web graphique aux utilisateurs d'administration afin qu'ils puissent gérer la gTSL. Il est développé avec les technologies HTML5, CSS3 et AngularJS. Dans le cadre de ce projet, nous avons décidé de réutiliser l'interface existante, appelée TL Manager. Cette interface permet notamment aux utilisateurs de créer des drafts qui peuvent ensuite être publiés dans la gTSL. Néanmoins, il n'est pas détaillé dans ce mémoire puisqu'il n'entre pas dans le cadre du stage.

Interface de programmation applicative REST

L'API [1] REST expose des web services fournissant toutes les opérations nécessaires à la gestion de la gTSL. Il a été développé en Java en s'appuyant sur le framework Spring. Cette interface de programmation encapsule les différents modules introduits en Section 5.4 permettant de gérer les données, d'authentifier les utilisateurs et de s'abonner aux listes de confiance. Elle est décomposée en trois parties comme présenté dans la Figure 6.2.

1. Une API, acronyme anglais de Application Programming Interface, est une interface de programmation applicative qui a pour but de fournir des services à d'autres logiciels.

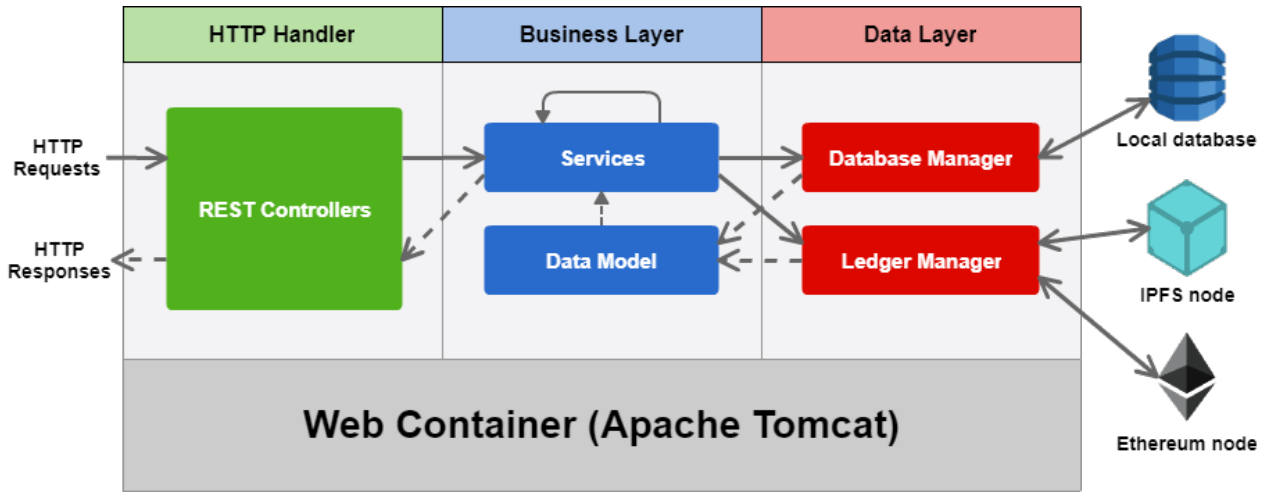


FIGURE 6.2 – Interface de programmation applicative

Le *HTTP Handler*[2] a pour rôle d'intercepter et de répondre aux requêtes HTTP. Il est composé de multiples contrôleurs chargés de faire appel à des services qui traitent les demandes des utilisateurs. Chaque contrôleur est défini sur une URL précise et expose des méthodes dans lesquelles sont configurés la méthode HTTP à utiliser ainsi que l'en-tête et le corps de la requête. Le *HTTP Handler* renvoie des réponses HTTP contenant un statut permettant de déterminer le résultat d'une requête ou d'indiquer au client une erreur. Si aucune erreur n'est survenue, le code de réponse HTTP 200 OK est envoyé.

La *Business Layer*[3] est la partie traitement des données de l'application. C'est dans cette partie que les données sont vérifiées, validées, transformées ou éventuellement rejetées si elles sont invalides. Elle est composée de services qui sont appelés par les contrôleurs REST. Les services font appel à la couche de données afin de lire, modifier ou écrire dans la base de données. Ils peuvent aussi faire appel à d'autres services si cela est nécessaire. Dans cette partie sont gérées les éventuelles erreurs et exceptions qui peuvent survenir lors du traitement des données.

La partie *Data Layer*[4] a pour objectif de gérer les données. La couche de données est composée d'une part du *Database Manager* et d'autre part du *Ledger Manager*. Le *Database Manager* a pour rôle de gérer les insertions, les mises à jour et les récupérations des données concernant les abonnements et les drafts en utilisant la base de données locale. Le *Ledger Manager* quant à lui joue le rôle d'interface entre le nœud IPFS et le nœud Ethereum, il a aussi à charge de gérer les données liées aux listes de confiance.

Nœud IPFS

Le nœud IPFS est connecté à un réseau pair-à-pair composé d'autres nœuds IPFS anonymes. Il a pour rôle de maintenir les données dans ce réseau en les répliquant à travers les autres nœuds IPFS appartenant au service de gTSL, ce qui forme ce que l'on appelle un cluster [5]. Le nœud interagit avec le système via le *Ledger Manager* qui est chargé de lui communiquer les données à stocker.

2. en français, gestionnaire HTTP.
3. en français, couche métier.
4. en français, couche de données.
5. Un cluster est un ensemble de serveurs sur un réseau qui agissent dans un but commun.

Nœud Ethereum

Le nœud Ethereum est connecté à un réseau pair-à-pair composé d'autres nœuds Ethereum anonymes. Il a pour rôle d'assurer l'intégrité des données grâce au consensus de la blockchain. Le nœud s'interface avec le *Ledger Manager* à qui il doit fournir l'état courant de la gTSL stocké dans un smart contract. L'état courant correspond à l'ensemble des hashes représentatifs de chaque liste de confiance. Son deuxième rôle est la gestion des utilisateurs. En effet, l'authentification est gérée par un smart contract qui maintient les clés publiques des personnes autorisées à modifier la gTSL.

Base de données

La base de données locale à chaque nœud a pour but de stocker les emails des personnes abonnées aux notifications ainsi que les drafts de listes de confiance. Les notifications sont donc gérées par le nœud auquel l'utilisateur a souscrit. L'intérêt de conserver les emails dans une base de données locale est de ne pas diffuser publiquement ces emails, ce qui serait le cas si ils étaient stockés dans la blockchain, afin de respecter la vie privée des utilisateurs. Concernant les drafts, on les stocke ni dans IPFS ni dans Ethereum mais bien dans une base de données locale. En effet, on ne veut pas rendre publique un draft puisqu'il peut être incomplet ou contenir des informations non vérifiées. De plus, il n'est pas pertinent de s'assurer de la validité des données d'un draft puisqu'il n'est pas rendu publique. Les utilisateurs d'administration de la gTSL doivent dans tous les cas déployer leur propre instance du nœud de gTSL comme expliqué en introduction de la Section 6.1, ils auront donc leur base de données dédiée contenant uniquement les drafts qu'ils ont établis.

6.1.2 Architecture globale

Le système présenté en Figure 6.1 sera déployé par tous les utilisateurs d'administration dans toute l'Europe et au-delà. En effet, chaque administrateur a besoin de son propre compte Ethereum, c'est-à-dire de son propre nœud Ethereum et de son propre identifiant de nœud IPFS, c'est-à-dire de son propre nœud IPFS. Cela permet d'augmenter le nombre de réplica des données et donc d'améliorer la résilience du système. De plus, il est envisagé de mettre en place des instances publiques afin que les utilisateurs externes puissent récupérer les données liées à la gTSL. Dans ce cas, l'interface d'administration n'est pas nécessaire. Ce système est open-source, cela implique que toute personne souhaitant déployer sa propre instance aura la possibilité de le faire. Malgré tout, il ne lui est pas possible de modifier la gTSL puisqu'il ne dispose pas des autorisations requises. Toutes ces instances forment ensemble un système décentralisé et distribué qui permet de maintenir l'entièreté des données.

La Figure 6.3 présente l'architecture globale du système, décentralisée et distribuée, composée de plusieurs nœuds de la gTSL. Certains composants ne jouant aucun rôle dans la décentralisation du modèle, ils ont été retirés des nœuds de gTSL dans un soucis de compréhension.

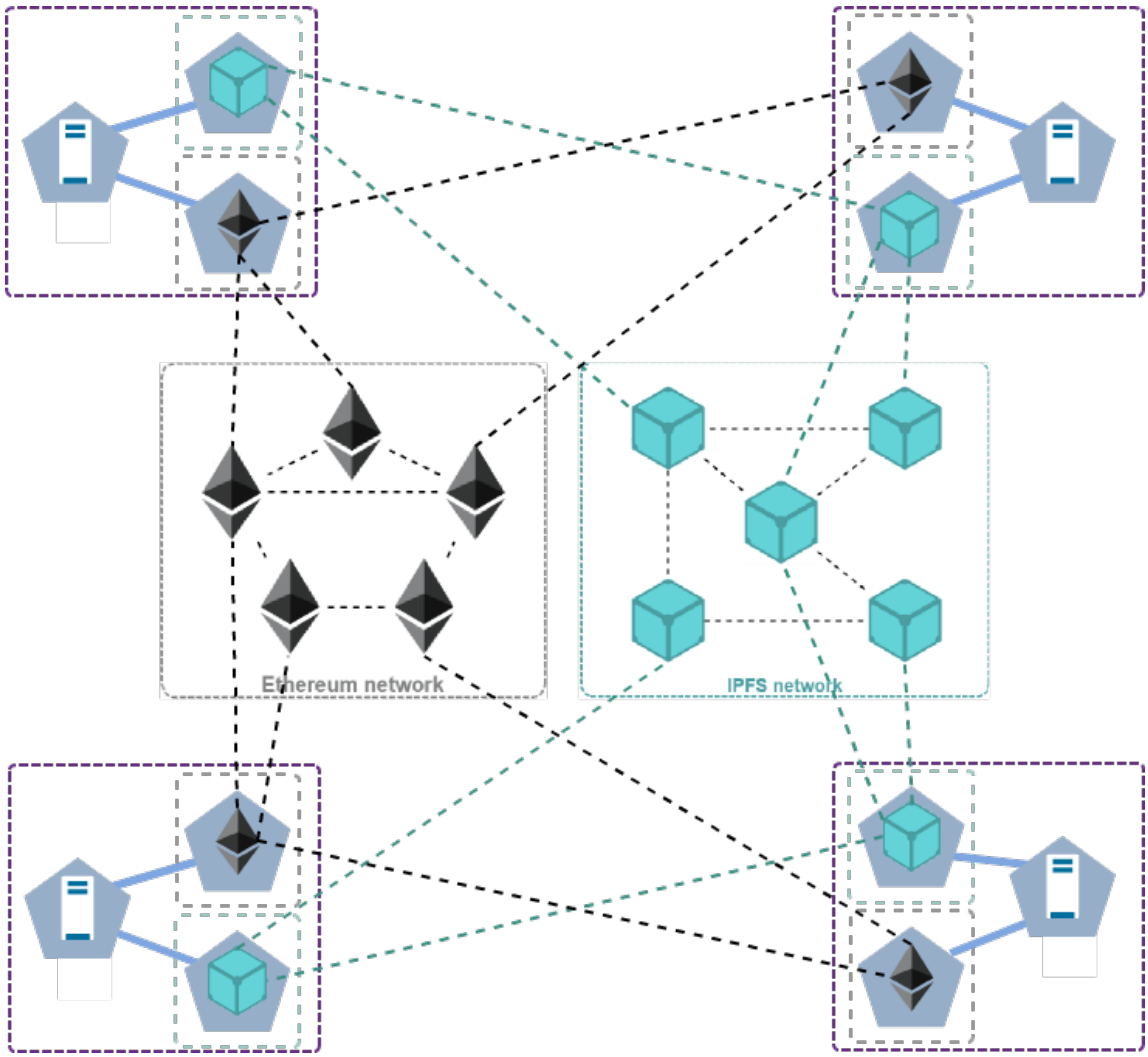


FIGURE 6.3 – Architecture globale

On observe dans la Figure 6.3 que la décentralisation se fait d'une part via le réseau IPFS qui maintient les données et d'autre part via le réseau Ethereum chargé de conserver l'état courant de la gTSL dans la blockchain. Il est aussi important de noter que ces réseaux sont publiques, ils existent donc d'autres nœuds anonymes, non engagés au niveau de la gTSL, mais qui participent tout de même à la résilience du système.

6.2 Implémentation des modules

Cette section présente les différents modules de la gTSL, et plus particulièrement les modules permettant de stocker les données, d'authentifier les utilisateurs, de gérer les notifications et de manipuler les listes de confiance.

6.2.1 Ledger Manager

Dans cette section est décrite la manière d'intégrer Ethereum et IPFS. Dans un premier temps est exposé l'intérêt d'utiliser ces technologies ensemble. Ensuite est présentée la structure de

données mise en place afin de les interfacer. Enfin est détaillé le système implémenté afin de gérer l'historique des versions de la gTSL.

Les explications fournies dans cette partie sont tirées du document Integration of Ethereum & IPFS disponible en Annexe A que j'ai rédigé dans le but de fournir une explication de l'utilisation de technologies décentralisées et distribuées dans le cadre du module de gTSL.

L'avantage de coupler IPFS & Ethereum

Un moyen simple de stocker les données de la gTSL est de les conserver dans une blockchain, cela évite la mise en place d'un système de fichiers décentralisé. Bien qu'une blockchain présente beaucoup d'avantages, elle présente également des inconvénients. L'un d'eux est que le déploiement d'un smart contract dans la blockchain Ethereum doit être payé. De plus, le prix augmente proportionnellement avec la taille du smart contract, c'est-à-dire son nombre d'opérations et la taille des données qu'il contient. Ainsi, le prix suit la quantité de données stockées. Une façon de réduire le coût est d'utiliser un réseau distribué afin de stocker les données gTSL et d'utiliser une blockchain afin de maintenir l'état des données. Dans IPFS, toutes les données sont adressées par leur hash, ce qui rend leur état facilement maintenable. Par conséquent, pour toutes les données d'une TSL, la blockchain ne maintient qu'un hash correspondant à l'état actuel de ces données. Cette solution réduit considérablement les coûts d'utilisation de la blockchain.

L'intégration des technologies

Comme expliqué dans la partie précédente, seul le hash des données est stocké dans la blockchain. Toutes les données réelles sont stockées dans IPFS, désigné comme étant la *couche de données*[6]. Un hash est l'adresse qui identifie une TSL donnée. Par conséquent, lorsqu'une TSL est stocké dans IPFS, un hash unique est automatiquement calculé à partir de ses données. Ce hash est ensuite stocké dans un smart contract dans lequel il est rattaché à un identifiant unique non mutable de la TSL, correspondant au code du pays de la TSL conformément au standard ETSI TS 119 612 [11]. Cet identifiant non mutable est ensuite utilisé par le système pour retrouver le hash d'une TSL. Contrairement à l'identifiant non mutable d'une TSL, le hash d'une TSL dépend des données qu'elle contient et est donc mutable. Plus particulièrement le hash est recalculé à chaque édition de la TSL par un État membre. La blockchain enregistre l'état courant des TSLs et met à jour l'état du contrat, on la désigne donc comme étant la *couche d'états*[7]. Le smart contract maintient l'état de chaque TSL et permet aux utilisateurs d'ajouter, de mettre à jour, de supprimer ou de récupérer une TSL. Il est important de noter qu'une autorisation est nécessaire pour appliquer des changements à une TSL mais que la lecture est disponible publiquement.

Une abstraction de la mise en œuvre du smart contract écrit en langage Solidity est fournie dans le Listing 6.1.

6. en anglais, Data layer
7. en anglais, State layer

```
contract Ledger {

  // Attributes
  mapping(bytes32 => bytes) public gtsl;

  function addTsl(bytes32 _code, bytes _hash)
    public onlyAuthorized
  {
    gtsl[_code] = _hash;
  }

  function updateTsl(bytes32 _code, bytes _hash)
    public onlyAuthorized
  {
    gtsl[_code] = _hash;
  }

  function removeTsl(bytes32 _code)
    public onlyAuthorized
  {
    delete gtsl[_code];
  }

}
```

Listing 6.1 – Ledger Contract

À noter qu'une fonction de récupération *get* est automatiquement générée par le compilateur pour chaque attribut. Le mot-clé *public* signifie que la fonction est accessible depuis l'extérieur du contrat, mais ne signifie pas que tous les utilisateurs ont la possibilité de l'exécuter. Le mot-clé *onlyAuthorized* définit que seul les personnes autorisées ont la possibilité de l'exécuter. La fonction *onlyAuthorized* est ce qu'on appelle un modificateur, il a pour rôle de filtrer l'exécution de la fonction en se basant sur les conditions d'exécution qu'il définit. Le modificateur *onlyAuthorized* est présenté dans le contrat de gestion des utilisateurs en Section 6.2.2.

Gestion des versions

Comme spécifié dans les besoins du service de gTSL, un historique de chaque TSL doit être conservé. Pour ce faire, l'idée est de maintenir une liste chaînée de toutes les versions de chaque TSL. Cette liste chaînée est stockée dans IPFS et chaque version, que l'on appellera *commit*, de la liste peut être considérée comme un objet JSON, défini dans le Listing 6.2.

```
{
  "dataAddress": "QmXXXXXXXXXXXXXX",
  "parent": "QmYYYYYYYYYYYYY",
  "author": "Ada Lovelace ada@lov.cc",
  "date": "Mon July 24 15:19:12",
  "signature": {"signature info"}
}
```

Listing 6.2 – Commit Object

Chaque attribut présent dans le Listing 6.2 est détaillé ci-dessous :

1. *dataAddress* est l'adresse (hash) IPFS pointant les données de la TSL pour la version actuelle ;
2. *parent* est l'adresse (hash) IPFS pointant la version (commit) précédente de la TSL ;
3. *author* identifie l'utilisateur qui a mis à jour la TSL ;
4. *date* représente l'horodatage du commit ;
5. *signature* sont des informations concernant la signature de l'utilisateur.

Au lieu de sauvegarder le hash des données dans la blockchain, le hash de l'objet de commit est sauvegardé. L'objet commit encapsule le hash des données, le hash de l'objet commit de la version précédente et d'autres informations complémentaires. Cela permet à l'utilisateur de récupérer les données de la TSL, de savoir qui est l'utilisateur qui l'a créé et le moment où elle a été créée, et d'accéder à la version précédente. Cette couche supplémentaire au dessus des données contenant des méta-données de la version est désignée comme étant la *couche de méta-données*[8].

Résultat

La Figure 6.4 résume la gestion de versions et l'intégration d'Ethereum & IPFS.

8. en anglais, Meta-data layer

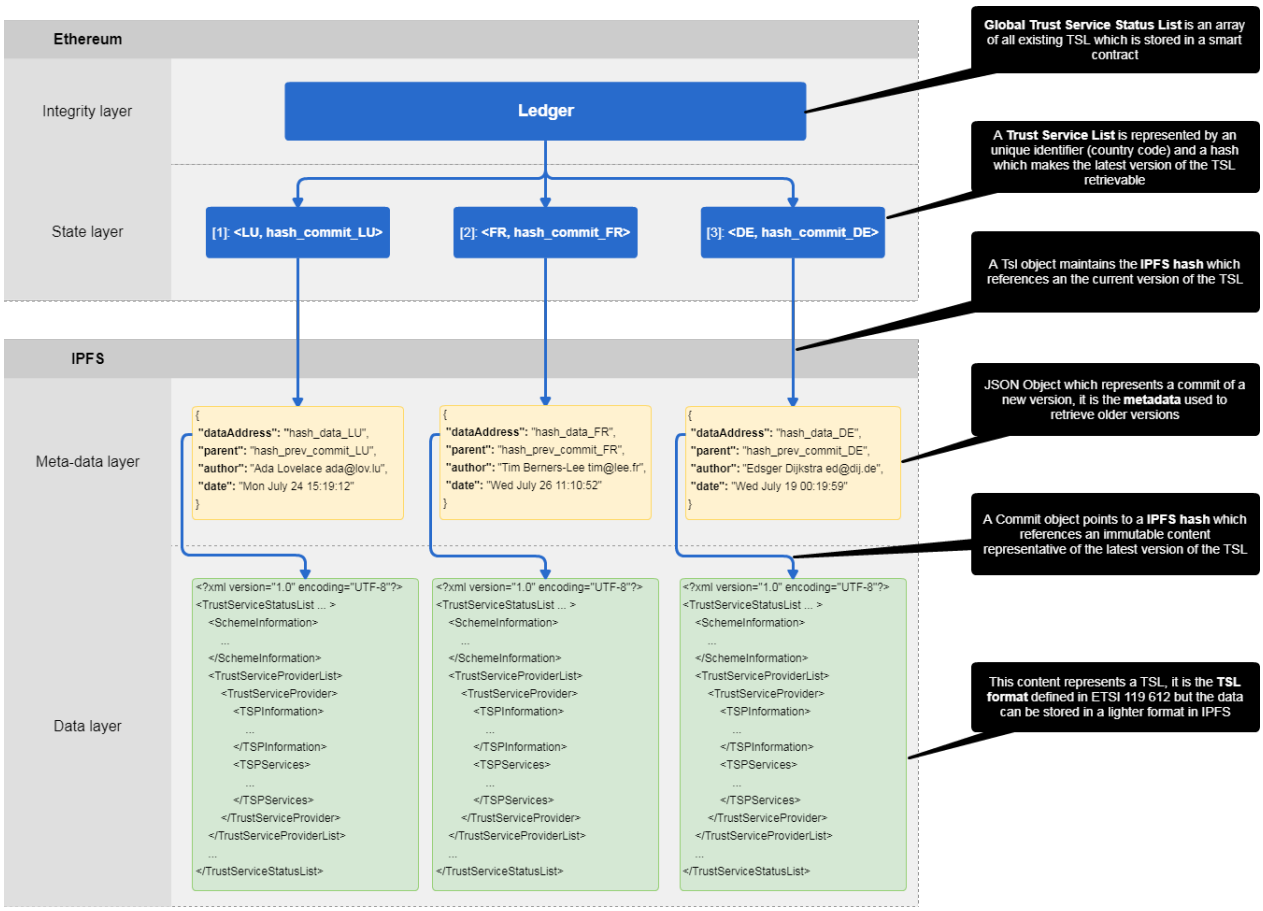


FIGURE 6.4 – Ledger Manager. Illustration extraite de *Integration of Ethereum & IPFS* [14]

6.2.2 Authentication Provider

Dans cette section sont décrits le service dédié à l'authentification et la gestion des utilisateurs. Dans un premier temps est exposé le processus d'authentification des utilisateurs. Ensuite est détaillé le processus de gestion des utilisateurs. Enfin est présenté le smart contract déployé dans la blockchain Ethereum, utilisé dans le but de réaliser ces actions.

Authentification des utilisateurs

L'authentification des utilisateurs est nécessaire uniquement pour les administrateurs de la gTSL. Le processus implémenté se détache du modèle classique d'authentification par nom d'utilisateur et mot de passe, qualifié d'authentification à un facteur. À la place, il repose sur une infrastructure à clés publiques qui permet d'authentifier les utilisateurs grâce à un certificat électronique, on qualifie cette authentification comme étant à deux facteurs. Afin de s'authentifier, l'utilisateur utilise sa clé privée, qui vérifiée contre sa clé publique permet de l'identifier.

Dans la pratique, l'authentification se fait via un smart contract déployé dans la blockchain Ethereum. Chaque utilisateur utilisant cette blockchain est identifié sur cette dernière grâce à une paire de clés. Dans Ethereum, et dans la technologie blockchain en général, lorsqu'un utilisateur émet une transaction, il la signe à l'aide de sa clé privée et fournit la clé publique dans le corps de la transaction. Cette clé publique peut être récupérée dans le smart contract, afin de répertorier les utilisateurs autorisés à procéder à des modifications de la gTSL. Le smart contract maintient

la liste des clés publiques des utilisateurs autorisées. Lorsqu'un utilisateur souhaite accéder à une fonctionnalité qui requiert une autorisation spécifique, le système va vérifier que sa clé publique est bien présente dans le smart contract de gestion des utilisateurs. Si sa clé publique est bien présente dans la liste des utilisateurs autorisés, alors le smart contract exécute la fonction appelée, sinon il refuse l'accès à cette fonction.

Gestion des utilisateurs

Cette section concerne uniquement les utilisateurs d'administration. En effet les utilisateurs externes ne sont pas concernés par l'authentification. La gestion des utilisateurs concerne le processus par lequel des administrateurs peuvent être ajoutés ou révoqués de la liste des utilisateurs autorisés à modifier la gTSL et à gérer les utilisateurs. En opposition avec un modèle classique, le modèle conçu ne repose pas sur un "super-administrateur" qui a le pouvoir de gérer les administrateurs.

Le système mis en place est basé sur un service de votes déployé dans la blockchain Ethereum. Les actions de gestion des utilisateurs reposent sur un consortium d'administrateurs ayant la possibilité de soumettre une proposition d'ajout ou de révocation d'un membre. Pour cela, l'utilisateur établit une proposition de votes, en précisant l'action à effectuer (ajout/suppression), le membre concerné par le vote ainsi que la date limite du vote. Pour une proposition donnée, chaque administrateur a la possibilité de voter en faveur ou en défaveur. En outre, tous les membres du consortium disposent du droit de vote et ce vote a le même poids pour chacun d'entre eux, il n'existe aucun privilège pour un votant.

L'avantage majeur de cette conception est que le système ne peut être altéré par un seul et unique utilisateur malhonnête, puisqu'il est nécessaire que la proposition soit validée par la majorité du consortium. Le seul moyen pour un attaquant de corrompre le système ou d'en prendre le contrôle est de réussir à être majoritaire dans le consortium. Pour cela, il faudrait soit qu'il réussisse à dérober les clés privées de plus de la moitié du consortium, soit qu'il réussisse à ajouter des clés publiques qu'ils détient dans la liste des clés autorisées, à hauteur d'au moins le nombre de clés déjà présentes dans la liste pour devenir majoritaire.

Afin de renforcer la sécurité du système et de le protéger des attaques Sybil, il est possible d'imaginer que l'on ne se base pas sur la majorité (50%) pour accepter la proposition mais sur un pourcentage plus élevé (80% par exemple). En effet, les actions d'ajout ou de suppression d'un administrateur du consortium sont décidées oralement entre les États membres, avant même que l'opération ne soit saisie dans le module de gTSL. L'intérêt d'augmenter la majorité d'une proposition est de protéger davantage le système. De plus, dans le but d'empêcher qu'un utilisateur puisse valider un vote tout seul, une proposition doit atteindre un quorum pour être validé. Ce quorum, tout comme le seuil de majorité, peut être augmenté afin d'améliorer la sécurité du système.

Pour une proposition, on différencie trois résultats possibles :
- succès, le consortium a approuvé la proposition car le quorum a été réuni, la majorité a voté en faveur de la proposition et la date limite est dépassée, le système procède donc à l'exécution de la proposition ;
- annulé, la date limite est dépassée mais le quorum n'a pas été réuni, le système abandonne l'exécution ;
- échec, la date limite est dépassée et le quorum a été réuni, mais la majorité a voté en défaveur de la proposition, le système abandonne l'exécution.

La Figure 6.5 détaille le processus mis en place dans le cadre d'une proposition de votes initiée par un administrateur.

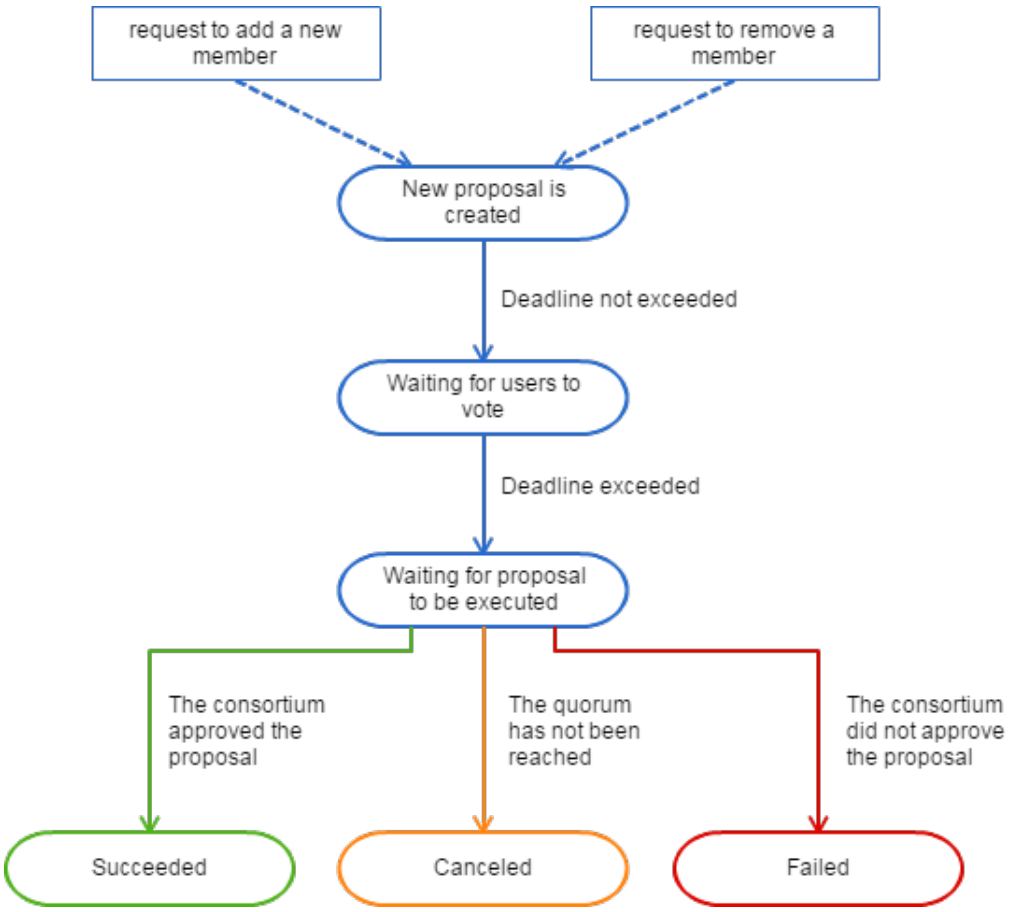


FIGURE 6.5 – Processus de gestion d'une proposition de votes

Implémentation

Une abstraction de la mise en œuvre du smart contract est fournie dans le Listing 6.3. Le code présenté a été simplifié dans un soucis de compréhension et ne représente pas l'ensemble du smart contract implémenté.

```
1  contract Consortium {
2
3    // Structures
4    struct Member {
5      bool authorized;
6    }
7    struct Proposal {
8      address member;
9      function (address) action;
10     mapping (address => bool) hasVoted;
11     bool[] votes;
12     bool result;
13     uint deadline;
14   }
```

```
15
16    // Attributes
17    mapping (address => Member) public members;
18    Proposal[] proposals;
19
20    // Modifiers
21    modifier onlyAuthorized {
22      require(members[msg.sender].authorized); _;
23    }
24
25    function requestAddMember(address _targetMember, uint _duration)
26      public onlyAuthorized returns (uint proposalID)
27    {
28      return newProposal(_targetMember, addMember, _duration);
29    }
30
31    function requestRemoveMember(address _targetMember, uint _duration)
32      public onlyAuthorized returns (uint proposalID)
33    {
34      return newProposal(_targetMember, removeMember, _duration);
35    }
36
37    function vote(uint _proposalID, bool _inSupport)
38      public onlyAuthorized
39    {
40      Proposal storage p = proposals[_proposalID];
41      require(now < p.deadline && !p.hasVoted[msg.sender]);
42      p.hasVoted[msg.sender] = true;
43      p.votes.push(_inSupport);
44    }
45
46    function executeProposal(uint _proposalID)
47      public onlyAuthorized returns (bool)
48    {
49      Proposal storage p = proposals[_proposalID];
50      require(now >= p.deadline);
51
52      /* minimum quorum not reached */
53      if (p.votes.length*100 < quorum) {
54        p.result = false;
55      } else {
56        /* marjority algorithm, not provided here*/
57        p.result = votesFor > votesAgainst;
58      }
59
60      /* execute action */
61      if (p.result) {
62        p.action(p.member);
63      }
64
65      return p.result;
66    }
67  }
```

Listing 6.3 – Consortium Contract

Les fonctions définies dans le Listing 6.3 sont détaillées ci-dessous :

- *onlyAuthorized* est un modificateur qui s'applique aux fonctions ayant pour but de vérifier que l'utilisateur appelant la fonction est répertorié dans la liste des administrateurs ;
- *requestAddMember* permet de créer une proposition d'ajout d'un nouvel administrateur ;
- *requestRemoveMember* permet de créer une proposition de révocation d'un administrateur ;
- *vote* permet aux utilisateurs d'administration d'appliquer leur vote à une proposition non expirée ;
- *executeProposal* permet de vérifier les votes d'une proposition expirée dans le but d'exécuter l'action associée si la proposition a été approuvée.

6.2.3 Subscription Provider

Le service d'abonnement possède trois fonctions qui sont l'abonnement, le désabonnement et la notification. Ce module est similaire à un service de newsletter [9] que proposent de nombreux sites marchands par exemple.

Abonnement

Tout utilisateur a la possibilité de souscrire au service d'abonnement de la gTSL. Pour cela, il lui suffit de fournir son adresse email et de sélectionner les pays pour lesquels il souhaite être notifié. Ces informations sont stockées par le système dans une base de données locale au nœud de souscription. Cela signifie que l'utilisateur est abonné à un nœud en particulier.

Désabonnement

Tout utilisateur abonné à une TSL a la possibilité de se désabonner du service de notifications. Pour cela, il lui suffit de cliquer sur le lien de désabonnement dans un des emails de notification qu'il a pu recevoir. Il sera alors désabonné de son nœud de souscription.

Notification

Le système a pour mission de notifier tous les abonnés aux TSLs. Lorsqu'une modification est effectuée, alors tous les nœuds en sont informés. Chaque nœud doit donc extraire dans la base de données la liste des personnes ayant souscrits à la TSL modifiée et se charge ensuite de les en informer par email.

9. Une lettre d'information, newsletter en anglais, est un document d'information transmis par courrier électronique à l'ensemble des abonnés qui y ont souscrit.

6.2.4 Lifecycle Manager

Le *Lifecycle Manager* a pour rôle de gérer le cycle de vie des TSLs. Il se charge de traiter, vérifier et valider les informations relatives aux TSLs, on dit qu'il applique la logique métier. Ce module permet d'une part de gérer les TSLs et d'autre part de gérer les drafts. Concernant les TSLs, il permet de traiter la validation de tous les champs de la TSL conformément au standard ETSI TS 119 612 [11] lors de l'ajout ou de l'édition. Il offre aussi la possibilité de procéder à des recherches sur les TSLs, d'importer une TSL au format XML et d'exporter une TSL au format XML. Les différentes techniques de recherche envisagées sur la gTSL sont évoquées dans le document gTSL Search Engine disponible en Annexe B. Concernant les drafts, il gère le processus de création et d'édition des drafts en indiquant pour chacun quels champs sont valides vis-à-vis du standard ETSI TS 119 612 [11]. Un draft peut être enregistré même s'il n'est pas valide, mais dans ce cas il ne pourra pas être publié dans la gTSL. À noter que les utilisateurs externes peuvent créer des drafts mais ne disposent pas des droits nécessaires pour publier les TSLs. Si un utilisateur externe est promu administrateur par la suite, il pourra tout de même publier les drafts réalisés en tant qu'utilisateur externe.

6.3 Présentation de la solution

Cette section concerne uniquement l'interface de programmation applicative. L'interface graphique d'administration a été réalisée dans la version antérieure du service de gestion des listes de confiance et sera réutilisée sans modifications majeures dans le cadre de la gTSL. Cette interface n'entre donc pas dans le cadre du stage et n'est pas présentée dans ce mémoire.

6.3.1 Interface de programmation applicative

L'interface de programmation applicative (API) de la gTSL correspond au module *Trust List Provider*. L'API implémentée respecte le modèle REST et se charge d'intercepter les requêtes HTTP des utilisateurs afin de les diriger vers le service adapté du *Lifecycle Manager*. Il correspond à l'ensemble des contrôleurs implémentés grâce au framework [10] Spring [11]. Cette section présente l'ensemble des contrôleurs définis dans l'implémentation du projet.

La Figure 6.6 définit l'ensemble des contrôleurs exposés par le *Trust List Provider*.

10. Un framework est un ensemble de composants développées par un tierce permettant de définir une architecture logicielle.

11. Voir https://spring.io/, sur ce site sont notamment disponibles des tutoriels d'utilisation du framework

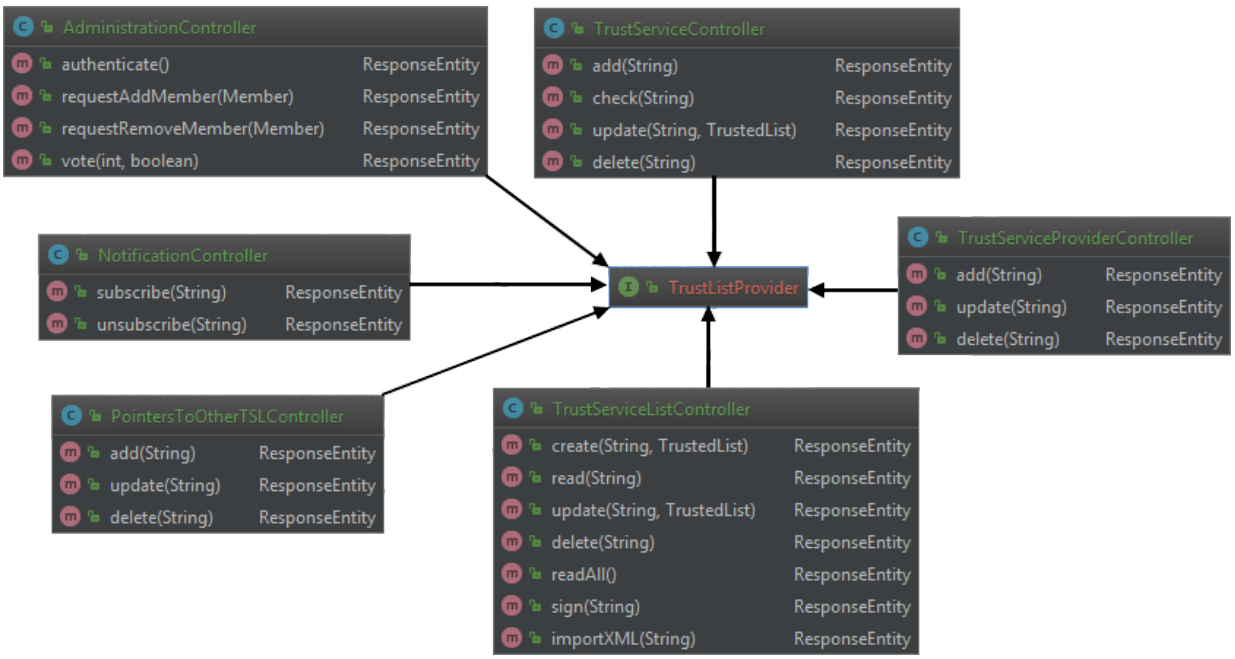


FIGURE 6.6 – Contrôleurs exposés par le Trust List Provider

Les contrôleurs définis dans la Figure 6.6 sont détaillés ci-dessous :

- Le contrôleur *Administration* permet aux administrateurs de s'authentifier pour ensuite pouvoir gérer les utilisateurs ;
- Le contrôleur *Notification* permet aux utilisateurs de s'abonner et de se désabonner des TSLs ;
- Le contrôleur *PointersToOtherTSL* permet aux administrateurs de gérer les pointeurs vers les autres TSLs ;
- Le contrôleur *TrustServiceList* permet à tous les utilisateurs de lire une TSL particulière ou toutes les TSLs, et permet aux administrateurs de créer, éditer, supprimer, signer ou importer une TSL ;
- Le contrôleur *TrustServiceProvider* permet aux administrateurs d'ajouter, éditer ou retirer un TSP d'une TSL.
- Le contrôleur *TrustService* permet aux administrateurs d'ajouter, éditer ou retirer un TS d'un TSP, et permet à tous les utilisateurs de vérifier un TS en fonction de critères de filtrage.

6.4 Validation de la solution

Durant ce stage, plusieurs méthodes de validation ont été mises en place. Étant donné que le projet est encore en réalisation au moment de la rédaction de ce mémoire, le processus de validation ne peut être considéré comme complet. Cependant, nous avons tout de même mis en place plusieurs mesures de vérification, dans le but d'attester de la conformité des travaux effectués avec les besoins initiaux présentés au Chapitre 3.

Dans un premier temps, la première validation a consisté à valider les choix opérés à la suite de l'état de l'art en début de stage. Cette étape a été très importante car c'est elle qui nous a permis de valider les technologies à utiliser dans le projet. Pour cela, j'ai réalisé une preuve de concept permettant d'interfacer les technologies Ethereum et IPFS dans le but de stocker des données dans un environnement décentralisé et distribué. Cette preuve de concept a ensuite fait l'objet

d'une démonstration à l'équipe de projet qui a pu valider l'utilisation des technologies sur la base du travail que j'ai réalisé.

Ensuite, d'autres points de validation ont eu lieu durant le stage. Tous les jours, une réunion rapide a été organisée afin de prendre connaissance du travail réalisé et restant pour chaque membre de l'équipe. Cette réunion qui s'inscrit dans la méthodologie Scrum permet de connaître l'avancement du projet pour chaque membre et permet aussi de mettre en avant les difficultés rencontrées ou de recevoir des validations orales et informelles dans nos choix d'implémentation.

Toujours dans la méthodologie Scrum, à chaque fin de sprint est organisée une réunion. Elle a pour but de faire le bilan des tâches réalisées durant les deux semaines de sprint et d'engager le nouveau sprint. Par ailleurs, elle permet de faire le point sur l'amélioration continue du processus de gestion de projet et de la réalisation des tâches afin de toujours augmenter la performance de l'équipe.

Une autre validation spécifique à ce projet est la conformité au standard ETSI TS 119 612 [11]. En effet, tout au long du stage il a été important et nécessaire de se référer régulièrement à ce standard puisque l'exigence première du projet est d'être conforme à celui-ci. Il a donc été nécessaire de faire valider chaque module en s'appuyant sur le standard lors de la phase de recherche, afin de s'assurer que la direction choisie permettait d'aboutir à une fonctionnalité valide.

Enfin, la validation a aussi porté sur l'aspect technique. Au fur et à mesure de l'avancée du projet, des tests unitaires et d'intégration ont été mis en place afin de valider les différentes fonctionnalités du service de gTSL.

7 Résultats obtenus & Perspectives

Les résultats définitifs du stage sont à l'heure actuelle inconnus puisque la date de livraison est prévue pour le 30 novembre 2017. Malgré cela, tous les retours du consortium d'entreprises du projet FutureTrust ont été jusqu'à présent positifs vis-à-vis du travail réalisé par l'équipe notamment concernant les documents de conception et les choix opérés. De plus, il y a un fort enthousiasme autour de la gTSL de par l'utilisation de la blockchain dans un projet européen.

Un autre point à soulever concernant ce projet est que c'est un projet de recherche et développement. Cela signifie que la Commission Européenne attend de l'innovation dans la conception et la réalisation tout en conservant une simplicité d'utilisation pour les utilisateurs finaux. Le challenge a donc été d'adapter une technologie récente et complexe qu'est la blockchain à un système existant. Notre objectif est que ce projet rencontre un grand succès afin qu'il soit adopté par la Commission Européenne pour remplacer l'architecture actuelle.

Un des résultats préliminaires du projet est la démonstration de notre capacité à utiliser la blockchain dans le cadre du projet FutureTrust suite à la preuve de concept que j'ai réalisée. Cette étape a eu lieu après que l'équipe ait validé les choix technologiques que j'ai proposés à la suite de l'état de l'art. À ce moment, le consortium a commencé à s'intéresser à notre conception de la gTSL et à sa mise en place imminente. J'ai donc rédigé un document explicatif de l'utilisation de la blockchain dans le cadre de la gTSL. Ce document s'intitule Integration of Ethereum & IPFS et est disponible en Annexe A. À la suite de la lecture de ce document, un membre de l'ETSI a montré un très grand intérêt pour notre travail et a invité l'équipe à venir présenter ce papier à un atelier blockchain qui aura lieu début 2018 au Luxembourg.

8 Conclusion

Ce stage de fin d'études vient conclure la formation de trois ans que j'ai suivie à Télécom Nancy. Cette formation m'a fait acquérir un grand nombre de compétences et connaissances techniques, lesquelles j'ai pu restituer tout au long de ce stage. Ces études m'ont aussi apporté des qualités humaines telles que la gestion du stress, l'évolution en équipe ou encore la logique de raisonnement, et cela notamment grâce aux nombreux projets qui ont été réalisés jusque maintenant.

Après avoir eu l'occasion d'étudier et d'exploiter la technologie blockchain pendant ce stage, je résumerais les défis de notre étude par les points suivants. Dans un premier temps, l'émergence de la blockchain a réellement bouleversé l'économie mondiale et a amené une prise de conscience générale des limites des systèmes centralisés. Aujourd'hui et plus que jamais, les entreprises sont intéressées par la mise en place d'architectures reposant sur la blockchain et plus particulièrement le secteur bancaire qui ne veut pas rater le virage des crypto-monnaies. Le problème persistant actuellement est le manque de régulation de cette technologie, plusieurs pays prévoient d'établir des réglementations autour de la blockchain et de la crypto-monnaie dans les années à venir.

En ce sens, l'initiative du projet FutureTrust quant à l'innovation autour de la confiance électronique à l'échelle mondiale est une excellente idée. Ce projet de recherche et développement ayant pour objectif de mettre en application le règlement de l'Union Européenne sur l'identification électronique et les services de confiance pour les transactions électroniques sécurisées au sein de l'UE a été une réelle opportunité de promouvoir la blockchain. En effet, l'Europe souhaite être un acteur majeur et un pionnier de l'utilisation de cette technologie. À ce titre, elle joue un rôle clé dans son développement. Malgré cela, une grande nécessité de communication autour de cette technologie persiste, car sa complexité peut freiner l'innovation et à terme entraîner sa disparition. Il est vrai qu'au premier abord, la blockchain est un concept flou et complexe car son fonctionnement est difficilement compréhensible pour un non initié à l'informatique et à la cryptographie. Le challenge pour les années futures se trouve dans la démocratisation de cette technologie par la mise en place de services grand public ergonomiques et simples d'utilisation. Le projet réalisé dans le cadre de ce stage propose une première approche d'intégration d'une blockchain au sein d'un service européen. De plus, l'engouement du consortium de FutureTrust, concernant les résultats déjà obtenus, est très encourageant pour la suite.

À titre personnel, ce projet a été un immense défi de par son innovation, sa complexité et sa portée. Sa réalisation a demandé de la réflexion, des remises en question et de la force de proposition car j'ai été amené à résoudre des problèmes techniques et scientifiques concernant les architectures décentralisées et l'intégration de la technologie blockchain. De plus, je suis convaincu que cette technologie est l'avenir du numérique et que les solutions à venir, particulièrement dans le secteur bancaire, vont créer une réelle rupture technologique. J'ai par ailleurs décidé de poursuivre ce projet dans l'entreprise Arηs afin de participer à l'élaboration du service de liste de confiance globale jusqu'à sa livraison et de continuer de concevoir des projets innovants autour de la blockchain.

Bibliographie / Webographie

[1] *RÈGLEMENT (UE) No 910/2014 DU PARLEMENT EUROPÉEN ET DU CONSEIL du 23 juillet 2014 sur l'identification électronique et les services de confiance pour les transactions électroniques au sein du marché intérieur et abrogeant la directive 1999/93/CE*, 2014. 4

[2] *Annual Report 2016.* ARHS, 2B Rue Nicolas Bové, 1253 Luxembourg, 2016. 3, 64

[3] Adam Back. *Hashcash - A Denial of Service Counter-Measure.* 2002. 20

[4] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System.* 2014. 32

[5] Blockgeeks. Smart contracts : The blockchain technology that will replace lawyers, 2017. 6, 64

[6] Blockgeeks. What is blockchain technology ?, 2017. 5, 64

[7] Vincent Bouckaert. *Design documentation - Global Trust Service Status List.* ARHS, 2B Rue Nicolas Bové, 1253 Luxembourg, 2017. 12, 18, 34, 39, 40, 41, 42, 43, 64

[8] Vitalik Buterin. *A Next-Generation Smart Contract and Decentralized Application Platform.* 2013. 31

[9] David Chaum. *Blind signatures for untraceable payments.* 1983. 20

[10] Wei Dai. *B-Money.* 1998. 20

[11] ETSI, 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE. *Electronic Signatures and Infrastructures (ESI); Trusted Lists*, 2016. 10, 11, 13, 14, 18, 42, 49, 57, 59

[12] Sunny King. *What is ppcoin ?* 2012. 21

[13] Satoshi Nakamoto. *Bitcoin : A Peer-to-Peer Electronic Cash System.* 2008. 1

[14] Yoann Raucoules. *Integration of Ethereum & IPFS*, 2017. 52, 64

Liste des illustrations

Listings

Acronymes

1. **API** - Application Programming Interface
2. **BFT** - Byzantine Fault Tolerant
3. **BPMN** - Business Process Model and Notation
4. **dApp** - decentralized Application
5. **eIDAS** - services de confiance pour les transactions électroniques sécurisées au sein de l'UE
6. **ESN** - Entreprise de Services du Numérique
7. **ETH** - ETHer
8. **ETSI** - European Telecommunications Standards Institute
9. **EVM** - Ethereum Virtual Machine
10. **gTSL** - Global Trust Service status List
11. **HTTP** - HyperText Transfer Protocol
12. **ICO** - Initial Coin Offering
13. **IPFS** - InterPlanetary File System
14. **JSON** - JavaScript Object Notation
15. **KSI** - Keyless Signature Infrastructure
16. **LoTL** - List of The Lists
17. **REST** - REpresentational State Transfer
18. **TS** - Trust Services
19. **TSL** - Trust Service Lists
20. **TSP** - Trust Service Providers
21. **UE** - Union Européenne
22. **UML** - Unified Modeling Language
23. **URI** - Uniform Resource Identifier
24. **URL** - Uniform Resource Locator
25. **XML** - eXtensible Markup Language

Glossaire

1. **attaque Sybil** : Une attaque Sybil est une attaque informatique qui vise à renverser un système par la création de fausses identités dans un réseau pair-à-pair.
2. **Bitcoin** : Bitcoin est une crypto-monnaie et un système de paiement pair-à-pair.
3. **cluster** : Un cluster est un ensemble de serveurs sur un réseau qui agissent dans un but commun.
4. **crypto-monnai**e : La crypto-monnaie aussi appelée monnaie cryptographique est une monnaie électronique basé sur les principes de la cryptographie.
5. **design pattern** : Un design pattern, appelé patron de conception en français, est un modèle considéré comme bonne pratique en réponse à un problème de conception d'un logiciel.
6. **framework** : Un framework est un ensemble de composants développées par un tierce permettant de définir une architecture logicielle.
7. **hash** : Un hash est le résultat d'une fonction de hachage qui permet d'identifier rapidement une donnée.
8. **logging** : Le logging est un dispositif permettant de stocker un historique des évènements attachés à un processus.
9. **minage** : Le minage est aussi un moyen de sécuriser le réseau en créant, vérifiant, publiant et propageant les blocs dans la blockchain.
10. **newsletter** : Une newsletter, appelé lettre d'information en français, est un document d'information transmis par courrier électronique à l'ensemble des abonnés qui y ont souscrit.
11. **pair-à-pair** : Le pair-à-pair est un modèle de réseau informatique similaire au modèle client-serveur mais où chaque client est aussi un serveur.
12. **point individuel de défaillance** : Un point individuel de défaillance, appelée single point of failure en anglais, est un point d'un système informatique dont le reste du système est dépendant et dont une panne entraîne l'arrêt complet du système.
13. **scrum meeting** : Un scrum meeting, communément appelé mêlée en français, est une réunion courte et énergique faisant partie de la méthodologie Scrum, qui permet à l'équipe de discuter de l'avancée d'un sprint et de lever les points bloquants du projet.
14. **smart contract** : Les smart contracts, appelés contrats intelligents en français, permettent d'échanger de l'argent, des biens, des parts ou n'importe quel actif de manière transparente, sans conflits et en évitant tout service intermédiaire.
15. **sprint** : Un sprint est une période sur laquelle sont réalisées des tâches définies.
16. **timestamping** : Le timestamping, appelé horodatage en français, est un mécanisme qui consiste à associer une date et une heure à un événement, une information ou une donnée informatique.
17. **Uber** : Uber est l'entreprise qui déclencha ce qu'on appelle l'ubérisation.

18. **ubérisation** : L'ubérisation est un phénomène économique désignant l'utilisation de services permettant aux professionnels et aux clients de se mettre en contact direct grâce à l'utilisation des nouvelles technologies.
19. **ubériser** : Ubériser est le verbe issu du substantif ubériser.
20. **web service** : Un web service correspond à l'implémentation d'une ressource identifiée par une URL.

Annexes

A Document de synthèse - Integration of Ethereum & IPFS

Integration of Ethereum & IPFS

Yoann Raucoules

ARHS Spikeseed

yoann.raucoules@arhs-spikeseed.com

July 25, 2017

Abstract

The present document provides an overview of the integration of Ethereum & IPFS in the context of the FutureTrust project. Those two technologies have been used in order to implement a Global Trust Service Status List (gTSL) in a decentralized and secure manner. The objective of using Ethereum & IPFS is to store the whole data regarding the gTSL in a distributed network. In this paper, we present an overview of each technology, the design of the adopted solution and the version control designed on top of IPFS which is used to keep track of the gTSL history.

I. Ethereum overview

Ethereum is an open blockchain platform that lets anyone build and use decentralized applications that run on blockchain technology. A blockchain is a distributed computing architecture where every network node executes and records the same transactions, which are grouped into blocks. Only one block can be added at a time, and every block contains a mathematical proof that verifies that it follows in sequence from the previous block. In this way, the blockchain's "distributed database" is kept in consensus across the whole network. Individual user interactions with the ledger (transactions) are secured by strong cryptography.

The particularity of the Ethereum blockchain is that it has been designed in order to enable users to write and run smart contracts. A smart contract describes a computer code that can facilitate the exchange of money, content, property, shares, or anything of value on the blockchain. When running on the blockchain, a smart contract becomes like a self-operating computer program that is automatically executed when specific conditions are met. Because smart contracts run on the blockchain, they run exactly as programmed without any possibility of censorship, downtime, fraud or third party interference. Ethereum's core innovation, the Ethereum Virtual Machine (EVM) is a Turing complete software that runs on the Ethereum network. It enables anyone to run any smart contracts, regardless of the programming language given enough time and memory. The most used language used on the Ethereum blockchain is Solidity which once compiled can be run by the EVM.

The great advantage of running applications on a blockchain is to benefit from its properties:

1. Immutability – A third party cannot make any changes to data.

2. Corruption & tamper proof – Apps are based on a network formed around the principle of consensus, making censorship impossible.

3. Secure – With no central point of failure and secured using cryptography, applications are well protected against hacking attacks and fraudulent activities.

4. Zero downtime – Apps never go down and can never be switched off.

II. IPFS overview

The InterPlanetary File System (IPFS) is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. It presents a new platform for writing and deploying applications, and a new system for distributing and versioning large data. In other words, IPFS provides a high throughput content-addressed block storage model, with content-addressed hyper links. This forms a generalized Merkle DAG, a data structure upon which one can build versioned file systems, blockchains, and even a Permanent Web. IPFS has no single point of failure, no nodes are privileged and nodes do not need to trust each other. IPFS nodes store IPFS objects in local storage. Nodes connect to each other and transfer objects. These objects represent files and other data structures.

The IPFS Protocol is divided into a stack of sub-protocols responsible for different functionalities:

1. Identities – manage node identity generation and verification.
2. Network – manages connections to other peers, uses various underlying network protocols.
3. Routing – maintains information to locate specific peers and objects. Responds to both local and remote queries.
4. Exchange – a novel block exchange protocol (BitSwap) that governs efficient block distribution. Modelled as a market, weakly incentivizes data replication.
5. Objects – a Merkle DAG of content-addressed immutable objects with links. Used to represent arbitrary datastructures, e.g. file hierarchies and communication systems.
6. Files – a file system hierarchy inspired by Git.
7. Naming – a self-certifying mutable name system.

These subsystems are not independent; they are integrated and leverage blended properties.

III. Design of the solution

In this part, we describe the way to integrate Ethereum and IPFS, and the purpose to use those technologies together.

i. The advantage of coupling the two technologies

A simple way to store the gTSL data is to keep them in a blockchain, this avoiding the need of a decentralized file system. Although a blockchain has a lot of advantages, it also have some drawbacks. One of them is that the deployment of a smart contract within the Ethereum blockchain has to be paid. Furthermore, the size of the smart contract (in terms of operations and memory) is directly linked to the price. Thus, the price follows the amount of data that are stored. A way to reduce the cost is to use a distributed network in order to store the gTSL data and use a blockchain in order to maintain the state of the data. In IPFS, all data are addressed by their hash, making their state easily maintainable. Hence, for each stored gTSL data, the blockchain should only maintain a hash corresponding to the current state of the data. This last solution reducing considerably our costs resulting to our use of the blockchain.

ii. The way to integrate the two technologies

As explained in the previous part, only the hash of the data is stored in the blockchain. All the actual data are stored in IPFS, which is called the *data layer*. A hash is the address which identifies a particular Trust Service List (TSL). Hence, when a TSL is stored in IPFS, a hash of its data is automatically computed. As expected, this hash is a unique identifier of the TSL. This hash is then stored in a smart contract in which it is attached

to the unique non-mutable identifier of the TSL (which is a country code according to [ETSI TS 119 612]). This non-mutable identifier can then be used by end users when they intent to find the hash of a TSL that they need. In contrast to the non mutable identifier of the TSL, the hash of a TSL depends on the data it contains, this making it a mutable identifier of its TSL. Hence, a TSL can be modified by a member state, and its hash must then be updated. As the blockchain saves the current state of the TSL and updating the TSL corresponds to modify the state of the contract, the blockchain is called the *state layer*. The smart contract maintains the state of each TSL and allows the users to add, update, remove or fetch a TSL. Note that an authorisation is required for modifying the TSL but reading is publicly available.

An abstraction of the smart contract's implementation is given in Listing 1:

Listing 1: *Ledger Contract*

```
contract Ledger {

  // Structures
  struct Tsl {
    bytes32 code;
    bytes hash;
  }

  // Attributes
  mapping(bytes32 => Tsl) public gtsl;

  function addTsl
    (bytes32 _code, bytes _hash)
    public
  {
    gtsl[_code] = Tsl({
      code: _code,
      hash: _hash
    });
  }

  function updateTsl
    (bytes32 _code, bytes _hash)
    public
  {
    gtsl[_code].hash = _hash;
  }

  function removeTsl
    (bytes32 _code)
    public
  {
    delete gtsl[_code];
  }

}
```

Note: A *get* function is generated by the compiler for every public attribute.

IV. Version control on top of IPFS

As specified in the gTSL's requirements, a history of each TSL must be kept. To achieve this, the idea is to maintain a linked-list of all the versions for each TSL. This linked-list is stored in IPFS and each node (so-called a *commit*) of the list can be viewed as a JSON object, defined as following :

Listing 2: *Commit Object*

```
{
 "dataAddress": "QmXXXXXXXXXXXXXX",
 "parent": "QmYYYYYYYYYYYYY",
 "author": "Ada Lovelace ada@lov.cc",
 "date": "Mon July 24 15:19:12",
 "signature": {"signature info"}
}
```

1. dataAddress – is the IPFS address (hash) pointing to the data of the TSL for the current version. This hash corresponds to the new address of the TSL.

2. parent – is the IPFS address (hash) pointing to the previous version (commit object) of the TSL.

3. author – identifies the user who updated the TSL.

4. date – represents the time-stamp of the commit.

Instead of saving the hash of the data in the blockchain, it is the hash of the commit object which is saved. This allows the user to retrieve: the data of the TSL; the user who

created it; the time at which it was created; and the previous version. This layer on top of the data containing meta-data of the version is called *meta-data layer*.

V. Results

This following diagram summarizes the version control and the integration of Ethereum & IPFS:

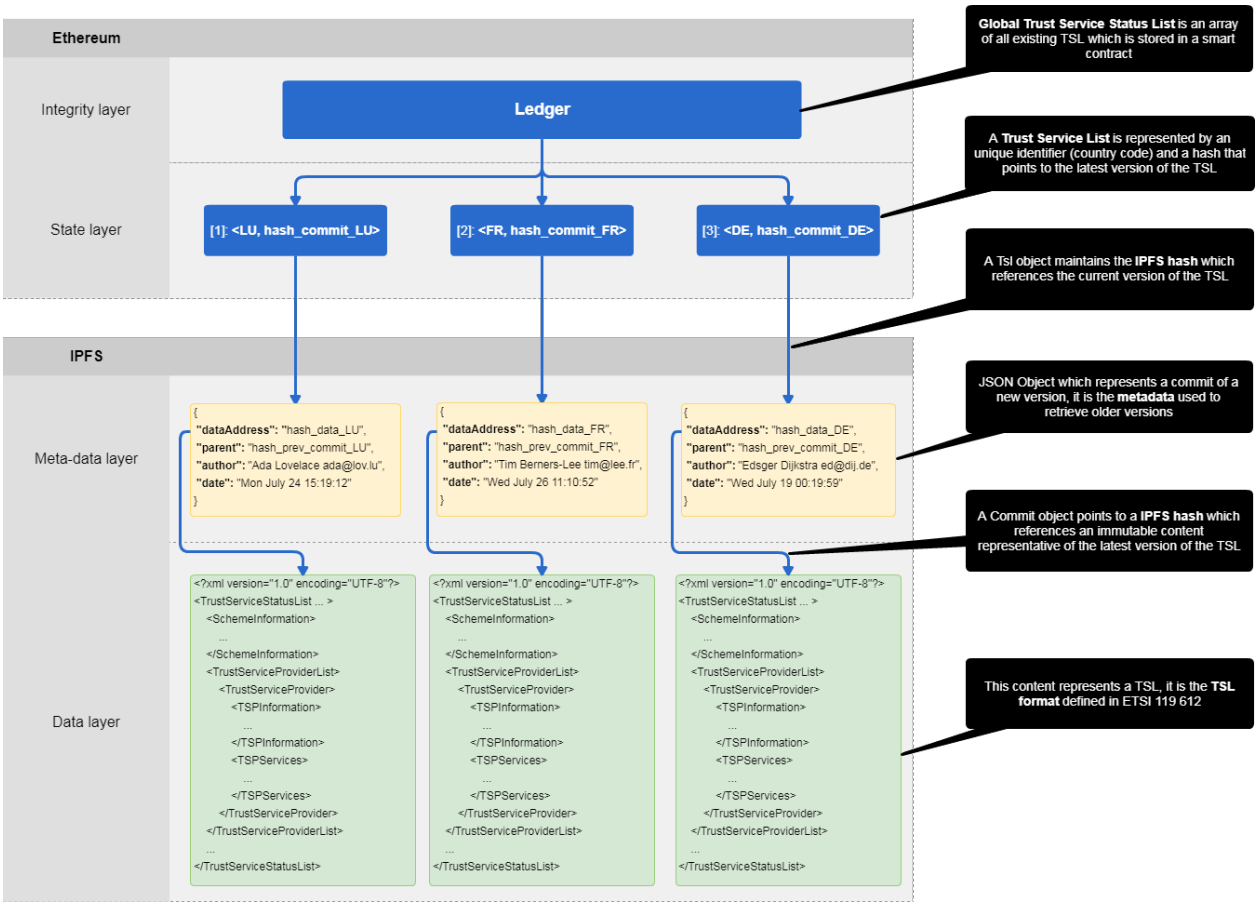


Figure 1: *Summary diagram*

References

[ETSI TS 119 612] Electronic Signatures and Infrastructures (ESI); Trusted Lists.

[Ethereum White Paper] A Next-Generation Smart Contract and Decentralized Application Platform.

[IPFS White Paper] Juan Benet (2014). IPFS - Content Addressed, Versioned, P2P File System.

B Analyse - gTSL Search Engine

FutureTrust

White Paper

gTSL Search Engine

Document Identification	
Date	13/07/2017
Status	Public
Version	Version 0.1

Abstract: The present document provides an overview of the search engine defined for the development of the Global Trust Service Status List that is part of the Future Trust project. It also specifies the technologies and methodologies used to implement the search operations.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

1. Executive Summary

The present document provides an overview of the search engine defined for the development of the Global Trust Service Status List that is part of the Future Trust project. It also specifies the technologies and methodologies used to implement the search operations.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

2. Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

3. About FutureTrust

Against the background of the regulation 2014/910/EU on electronic identification (eID) and trusted services for electronic transactions in the internal market (eIDAS), the FutureTrust project, which is funded within the EU Framework Programme for Research and Innovation (Horizon 2020) under Grant Agreement No. 700542, aims at supporting the practical implementation of the regulation in Europe and beyond.

For this purpose, the FutureTrust project aims to address the need for globally interoperable solutions through basic research with respect to the foundations of trust and trustworthiness, actively support the standardisation process in relevant areas, and provide Open Source software components and trustworthy services which will ease the use of eID and electronic signature technology in real world applications. In particular, the FutureTrust project will extend the existing European Trust Service Status List (TSL) infrastructure towards a “Global Trust List”, develop a comprehensive Open Source Validation Service as well as a scalable Preservation Service for electronic signatures and seals and will provide components for the eID-based application for qualified certificates across borders, and for the trustworthy creation of remote signatures and seals in a mobile environment.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

4. List of the different approaches

4.1 Use orbit-db

4.1.1 Description

orbit-db is a serverless, distibuted, peer-to-peer database. It has been developed on top of IPFS. Data in orbit-db can be stored in a Key-Value, Eventlog, Feed, Documents or Counters store. The implementation is written in Node.js.

4.1.2 Advantages

The main advantage is that orbit-db is developed on top of IPFS, the technology use in the project in order to store the data.

4.1.3 Disadvantages

orbit-db can only be used in a Javascript environment. It cannot provide a way to manage the different versions of the gTSL. The implementation is not optimized, it is based on a Logger in which each data is appended to the end of a JSON file which is stored in IPFS. It means that you need to load the whole file to perform actions on the database. Furthermore, orbit-db don't use indexes to retrieve data in an optimized way and there is no concurrency control.

4.2 Use aolog

4.2.1 Description

aolog is an append only log on IPFS with indexing. It is implemented in Javascript.

4.2.2 Advantages

It is based on IPFS and uses a Finger Tree as a data structure and a Bloom filter as a search method.

4.2.3 Disadvantages

It is implemented using append-only log storage, it means that it is impossible to update or remove data. Futhermore, it does not take into account the structure of the data, it is only lines containing data without links between those lines, so it impossible to deal with specific data structure.

4.3 Use YaCy

4.3.1 Description

YaCy is a distributed search engine that anyone can use to build a search portal for thier intranet or to help search the public internet. It is implemented in Java.

4.3.2 Advantages

It is a distributed search engine and anyone can deploy an instance in order to grow up the network of the search engine.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

4.3.3 Disadvantages

YaCy cannot be used on top of IPFS, unless if we modify the code to adapt it to IPFS. Futhermore, it only allow full-text search in the whole referenced pages and does not take into account the data structure used in the gTSL implementation.

4.4 Use Elasticsearch

4.4.1 Description

Elasticsearch is a distributed, RESTful search and analytics engine. It performs indexes on data and indexes are stored in JSON format. It is queryable and have a lot of options.

4.4.2 Advantages

The search operations are performed quickly using indexes. The queries can be customized depending on the data.

4.4.3 Disadvantages

Elasticsearch stores all the indexes locally, it means that we need to have a copy of an indexed gTSL. Elasticsearch can be distributed between multiple nodes but this increases the complexity of the application. Futhermore, it is needed to synchronize all Elasticsearch instance when CRUD operations are performed on the gTSL.

4.5 Reimplement an existing database on top of IPFS

4.5.1 Description

The idea is to use an existing implementation of a open-source database such as MongoDB and adapt it to allow data to be stored in IPFS.

4.5.2 Advantages

This is the most optimized way to perform search operations and store data. All data are stored in IPFS but on top of it there is a layer which allows us to query those data.

4.5.3 Disadvantages

It is very complicated to implement such a database, so it will take months or years and it is not the purpose of the project.

4.6 Implement a search engine based on ipfs-search

4.6.1 Description

ipfs-search is a search engine for IPFS. It performs searches on the whole IPFS public network. It is implemented in Node.js and Go.

4.6.2 Advantages

ipfs-search uses Elasticsearch to index data. It can be a starting point to implement a search engine based on Elasticsearch and IPFS.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

4.6.3 Disadvantages

ipfs-search only allows full-text search and does not take into account the file type. It means that a search can be performed in a JSON file as well as an image. The problem which can be raised is that we need to distribute all indexes between the nodes and to synchronize all instances of Elasticsearch.

4.7 Implement a custom graph-based search engine

4.7.1 Description

The idea is to represent the gTSL as a graph and perform search operations throughout this graph using well-known algorithms. Each node of the graph will represent a specific piece of data (chunks) which can be a Trusted List, a Trust Service Provider or a Trust Service. A chunk is not stored as the data itself but only its hash representation is stored which makes the whole graph lighter.

4.7.2 Advantages

Use hashes make the algorithm more efficient because if a data is not interesting the chunk will not be loaded in memory so only relevant data are kept in memory. This representation allows parallel actions when searching. It is possible to deal with different versions just by building the graph corresponding to the choosen version.

4.7.3 Disadvantages

The implementation has to be designed from scratch, even if existing algorithms from graph theory can be used. Another inconvenient is that the code will not be reusable entirely for other use cases because it will be optimized for the gTSL.

4.8 Implement a custom search engine based on inverted index

4.8.1 Description

The idea is to index data from the gTSL using inverted indexes. The purpose of this technique is to reference a set of values for a specific. For example, if we want to filter by countries (or Trusted Lists), we can register a hash table in which the user provide the country and the engine returns the set of Trust Service Providers attached to this country.

France	{ANTS, Caisse des dépôts, CertEurope, La Poste, Ministère de la Justice...}
Luxembourg	{LuxTrust}
Belgium	{Certipost, Zetes, QuoVadis BVBA}

You can optimize the following by only store hash representation of the data.

4.8.2 Advantages

It allows searches to be performed quickly because retrieving data is performed in a complexity of O(1).

4.8.3 Disadvantages

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

It is needed to store indexes in IPFS and maintain the hashes of the current in a smart-contract. This increases the complexity of adding, updating and removing data. The indexes have to be synchronize between all nodes to be efficiently used and concurrency control have to be implemented. Furthermore, this means to define a specific data structure in order to perform searches. It is mandatory to find a way to deal with versions. Another inconvenient is that the code will not be reusable entirely for other use cases because it will be optimized for the gTSL.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

5. References

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

C Documentation - gTSL - Project Setup Documentation

GTSL - Project Setup Documentation

This page provides an overview of the setup implemented for the development of the Global Trust Service Status List (GTSL) that is part of the Future Trust project. It also specifies the technologies and methodologies used to implement the setup.

Tools overview

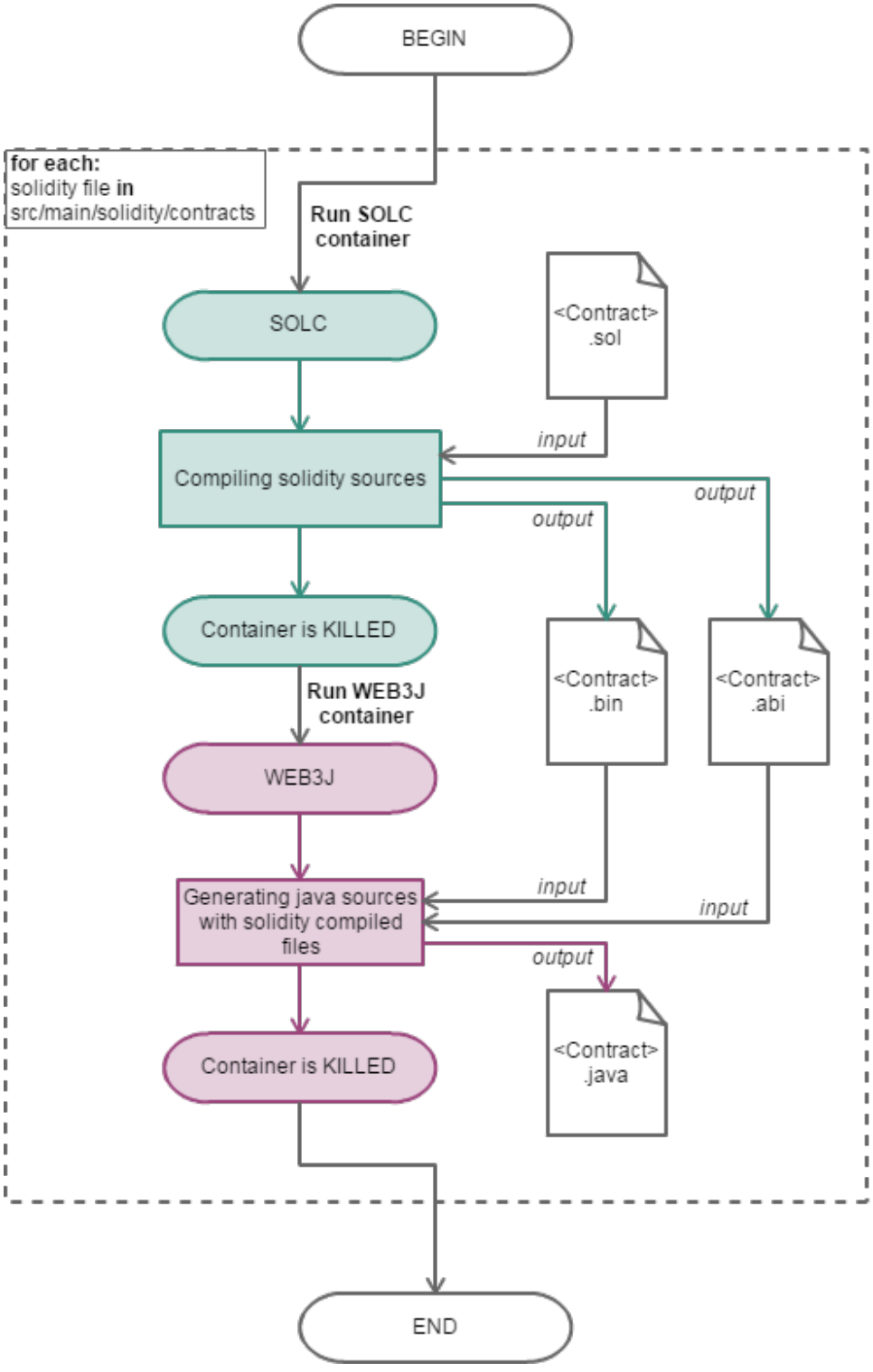
Tool	Version	Host	Description
Java	1.8	Local	programming language
Solidity	0.4.11	None (configured in solidity file itself)	smart-contracts language
Spring Boot	1.5.4.RELEASE	None (configured in pom.xml)	framework designed to simplify the development of Spring applications
Maven	3.5.0	Local	software project management and comprehension tool
Docker	17.05.0-ce	Local	open platform to build, ship, and run distributed applications
Docker-compose	1.13.0	Local	tool for defining and running multi-container Docker applications
Docker-machine	0.11.0	Local (required only on Windows & Mac OS)	tool that lets you install Docker Engine on virtual hosts
Ethereum	v1.6.5	Docker container	decentralized platform based on a blockchain that runs smart contracts
IPFS	v0.4.9	Docker container	protocol designed to create a permanent and decentralized file system
Solc	stable	Docker container	Solidity compiler
Web3j Command Line Tool	v2.2.1	Docker container	tool for generating Java sources from Solidity binaries

Project setup

The GTSL uses Ethereum as a ledger to keep track of the state of each trusted list it contains. On the other hand, the GTSL uses IPFS in order to store data regarding each trusted list. The application itself is a Spring Boot application.

Build smart-contracts

In the project, smart-contracts are written in Solidity and will be used to define the behaviour of the ledger. In order to use those smart-contracts in the application, it is needed that they are generated as Java classes. To achieve this, a process of compiling and generating has been implemented as described in the figure below. The implementation is a bash script that uses Docker containers for the Solc compiler and the Web3j generator.

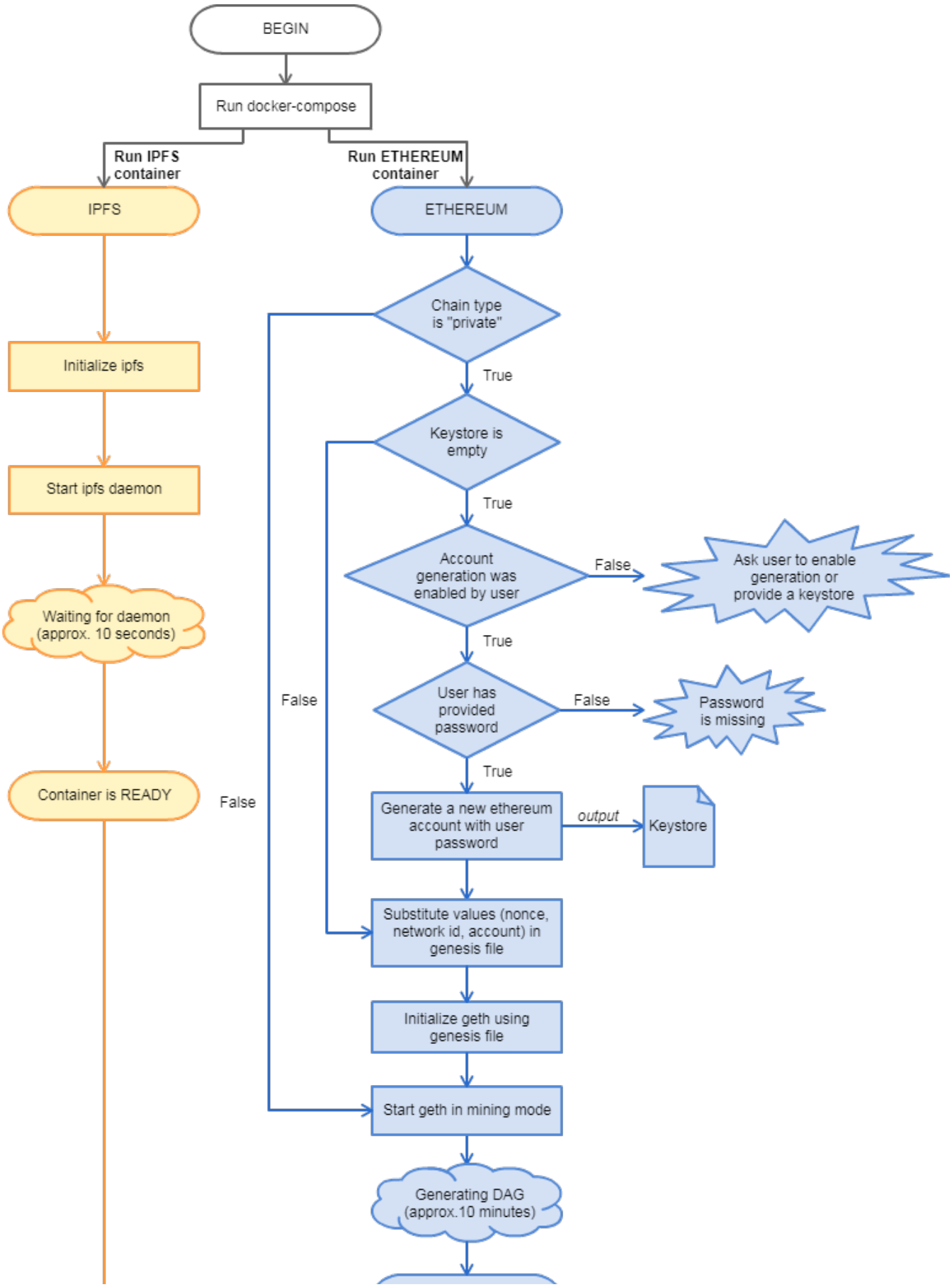


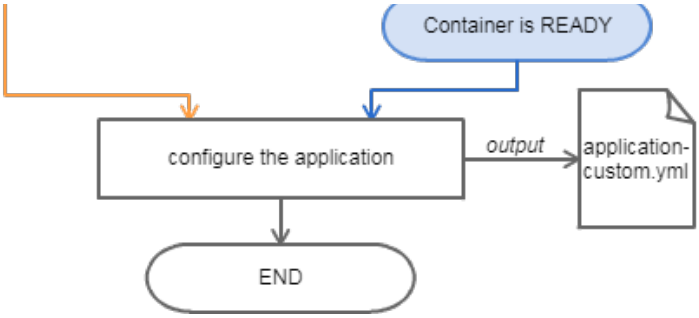
The process is divided into two parts and both parts are applied to all solidity files. First, the script runs a SOLC container that takes as input the solidity source file and creates a binary file and a .abi file which described the smart-contract. Then, those two files are used as input for the WEB3J container that generates a java source file for the smart-contract.

Warning: Only the smart-contracts in the folder *src/main/solidity/contracts* are generated. Those smart-contracts are "*final*" smart-contracts (final in the sense that there is no smart-contract which extends it). If you define *abstract* contracts, you have to put them in a sub-folder.

Run the dependencies

For the application to work it is needed that an IPFS node and an Ethereum node are up. The process described in the figure below explains how it is achieved using Docker containers. Before the application starts, the two containers have to be ready. It means an IPFS daemon is started and a private Ethereum blockchain is generated in mining mode. Then, some files are generated into the **outputs** folder. Those files have to be copied into the **gtsl** project.


BEGIN
Run docker-compose
Run IPFS container
Run ETHEREUM container
IPFS
ETHEREUM
Initialize ipfs
Start ipfs daemon
Waiting for daemon (approx. 10 seconds)
Container is READY
False
Chain type is "private"
True
Keystore is empty
True
Account generation was enabled by user
False
Ask user to enable generation or provide a keystore
True
User has provided password
False
Password is missing
False
True
Generate a new ethereum account with user password
output
Keystore
Substitute values (nonce, network id, account) in genesis file
Initialize geth using genesis file
Start geth in mining mode
Generating DAG (approx.10 minutes)


Container is READY
configure the application
output
application-custom.yml
END

D Documentation - gTSL - Project Setup Guidelines

GTSL - Project Setup Guidelines

This page provides the guidelines for setting up the Global Trust Service Status List (GTSL) project that is part of the Future Trust project.

Prerequisites

Tool	Version	Host	Description
Java	1.8	Local	programming language
Git	latest	Local	versioning control system
Maven	latest	Local	software project management and comprehension tool
Docker	latest	Local	open platform to build, ship, and run distributed applications
Docker-compose	latest	Local	tool for defining and running multi-container Docker applications
Docker-machine	latest	Local (required only on Windows & Mac OS)	tool that lets you install Docker Engine on virtual hosts

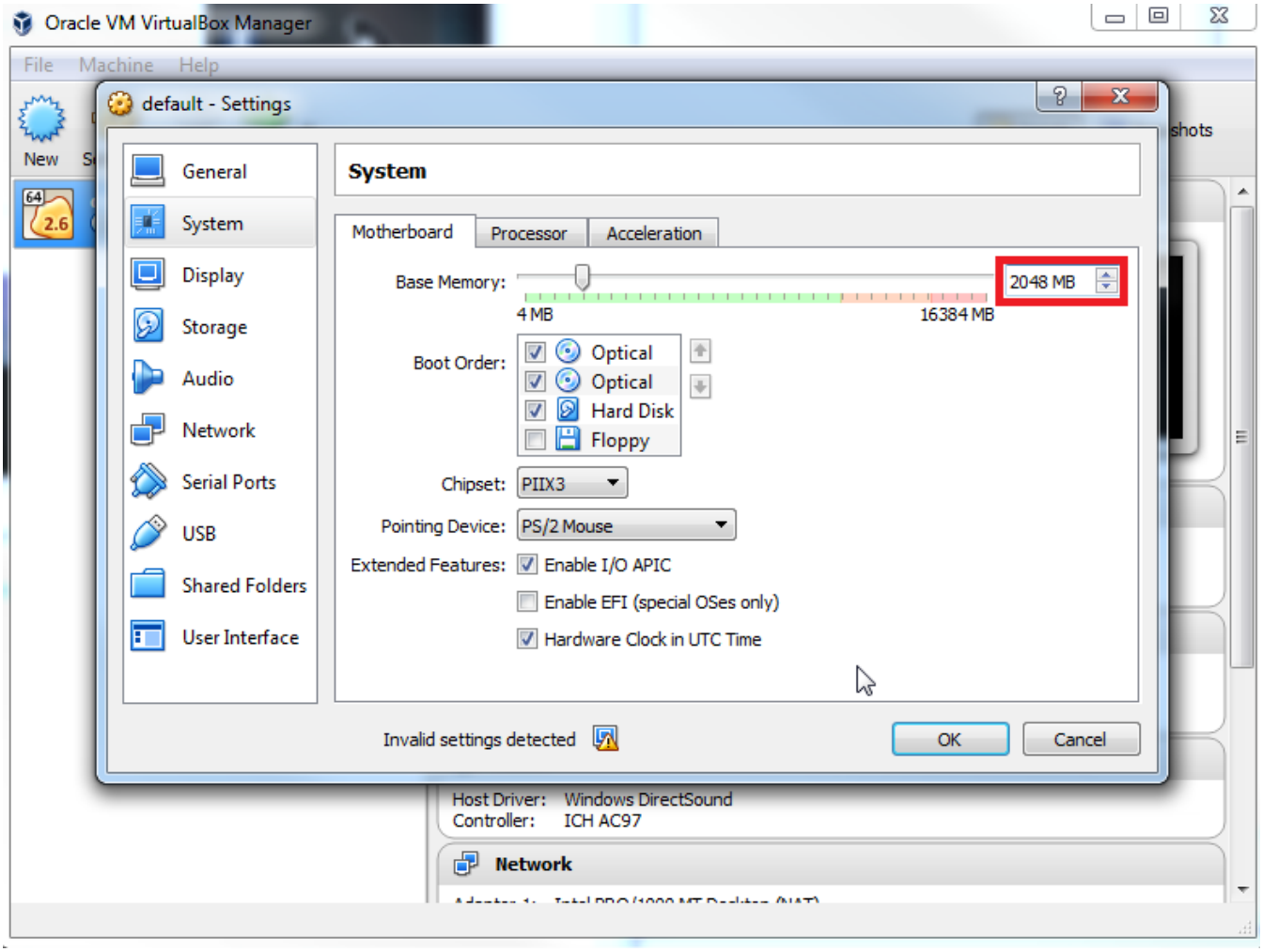
Warning

On Windows and Mac OS, you have to clone all the repositories (provided later in the document) wherever you want **in your HOME**. The docker feature which allows you mounting volumes only works in your HOME.

If you are using *docker-machine* (required on Windows and Mac OS while using *Docker Toolbox*):

- make sure to **always use the *Docker Quickstart Terminal*** (provided with the *Docker Toolbox);*
- make sure that your virtual machine has a **minimum 2 Go** of RAM, see explanations below.

If you are using Docker Toolbox, then open VirtualBox, shutdown the docker-machine (should be **default**), then right-click on it, Settings... System and update the memory value (see the screenshot below).



Build smart-contracts

> **If you don't plan to modify the Solidity smart-contracts, please skip this part.**

First of all, clone the repository of the project on your machine :

Bash

```
git clone
https://<username>@gitlab.arhs-developments.com/FutureTrust/gtsl-ethereum.
git
cd gtsl-ethereum
```

Compile Solidity sources and generate Java classes

Building smart-contracts is the process that allows us to generate the Java classes corresponding to the Solidity contracts defined in *src/main/solidity/contracts*.

Warning: Java files which will be generated are not versioned by Intellij IDEA, you have to add them to Git manually using the *git add* command.

For building smart-contracts, simply use the following command :

```
Bash
./src/main/scripts/build.sh
```

> ***TEMPORARY:*** *For the moment, this project cannot be used as a Maven dependency in the main project* ***gtsl*** *because the Nexus is not available yet. So, in order to use the generated Java files in the main project please Copy/Paste Java files in the* ***gtsl*** *project.*

Clean containers, images and temporary files

You can clean the project by running the following command :

```
Bash
./src/main/scripts/clean.sh
```

This command removes the temporary files, containers and the images used in the build process.

Run the dependencies

First of all, clone the repository of the project on your machine :

```
Bash
git clone
https://<username>@gitlab.arhs-developments.com/FutureTrust/gtsl-docker.git
cd gtsl-docker
```

Manage your Ethereum account

Before starting the application, you have to manage your Ethereum account.

- If you already have an account, you can use it by copying the keystore file into **src/main/docker/ethereum/data/keystore/.**
- If you do not have an account the script will generate one for you.

You have to provide the password of the account in order to load your credentials in the application.

To do this, create a file named **password** into **src/main/docker/ethereum/src/** and write the password of your account **on the first line as a plain text with no blanks**.

- If you already have an account, the password must be the password of the account you provide.
- If you do not have an account, the password will be used to generate a new account and will be the password of the account.

Note: For development, use a trivial password ("123456" for instance), the application will never ask you for this password.

Warning: Beware not to commit your password or keystore on the git repository, even if those files are part of the *.gitignore*.

Run the starting script

If you are using *docker-machine* (which should be the case on Windows and Mac OS), pass in as arguments the machine name (if you are using Docker Toolbox the machine name should be **default**). The endpoints will be reached through the *docker-machine*.

If you don't use *docker-machine*, don't pass any arguments. The endpoints will be reached on *localhost*.

The following command will run the dependencies.

Bash

```
./src/main/scripts/start.sh <docker-machine>
```

You have to wait for containers are ready, it should take approximately 10 minutes.

When the dependencies are ready, you have to Copy/Paste the files located in the *outputs* directory into the **gtsl** project. How to do this is described in the next part "Start the application".

Note: If you already did the configuration above and you run the starting script again, the script will use the keystore which was generated previously. It means that you don't have to Copy/Paste outputs files every time you run the starting script.

Clean containers and images

Warning: This command will remove all containers, even if they are running, including the containers not related to the project.

You can clean the project by running the following command :

Bash

```
./scripts/clean.sh
```

Start the application

First of all, clone the repository of the project on your machine :

Bash

```
git clone
https://<username>@gitlab.arhs-developments.com/FutureTrust/gtsl.git
cd gtsl
git checkout develop
```

Before starting the application, you need to provide your keystore and your configuration files. Those files had been generated in the previous part "Run the dependencies" and are located in the *outputs* directory in the **gtsl-docker** project. Copy and paste, the keystore and the configuration files into **gtsl-web/src/main/resources** in the **gtsl** project.

Then, you can start the application using the following command.

Bash

```
./scripts/start.sh
```

The Spring Boot application should be running.

The application is running on http://localhost:8081/.

Troubleshooting

To know if containers work well, use the *docker logs* command :

Bash

```
docker logs <container>
```

List of containers :

- ipfs-node
- ethereum-node

E Analyse - Comparaison des technologies

10/05/2017	OrbitDB	BigchainDB	Openchain	Swarm	IPFS	Storj	Factom	Monax	Ethereum
main criterion	serverless, distributed, peer-to-peer database on top of IPFS	scalable blockchain database	distributed ledger	distributed storage platform and content distribution service	peer-to-peer hypermedia protocol	distributed, encrypted, and blazing fast object storage peer-to-peer cloud storage network	distributed, decentralized protocol running on top of Bitcoin that handles data by providing an unalterable records	open platform for developers and devops to build, ship, and run blockchain-based applications	decentralized platform based on blockchain that runs smart contracts
description	**not well documented** uses IPFS as its data storage and IPFS Pubsub to automatically sync databases with peers	database with added blockchain characteristics - digital assets way.	issue and manage digital assets in a robust, secure and scalable	a native base layer service of the ethereum web 3 stack. Swarm provides a decentralized and redundant store of Ethereum's public record. peer-to-peer storage and serving solution	IPFS (the InterPlanetary File System) aims to replace HTTP IPFS is a new hypermedia distribution protocol, addressed by content and identities. IPFS enables the creation of completely distributed applications. IPFS is a distributed file system that seeks to connect all computing devices with the	**documentation only covers "How to use Storj"** **there is no explanation on basic concepts; there is only a blog**	It allows users to write data to its ledger for a small fee. The Factom blockchain is orders of magnitude less expensive and has orders of magnitude more capacity for transaction volume than Bitcoin.	The Monax platform is for building, testing, maintaining, and operating ecosystem applications with a blockchain backend Monax gives access to developers and DevOps who use blockchains, smart contracts, key management systems, and distributed data lakes to a core set of tools tackling the	Ethereum is a decentralized platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference.
maturity	**Starting date** : December 2015 **1st release** : Unknown (seems to be deleted) 6 contributors 282 github stars	**Starting date** : Summer 2015 **1st release** : 10/02/2017 38 contributors 1.1k github stars release frequence > 1 per month	**Launching date** : October 2015 **1st release** : 06/09/2015 1 contributor 200 github stars release frequence = 1 every two months until December 2016	**Starting date** : January 2015 122 contributors on geth 3.8k github stars on geth release frequence > 1 per month	**Starting date** : July 2014 **1st release** : 27/02/2015 121 contributors on go-ipfs 2.9k github stars on go-ipfs release frequence > 1 per month	**Starting date** : June 2014 **1st release** : 25/02/2016 22 contributors 216 github stars release frequence > 3 per month	**Starting date** : May 2013 **1st (known) release** : 09/01/2017 (v0.4.0.0) 46 contributors 52 github stars release frequence > 1 per month	**Starting date** : October 2014 **1st (known) release** : 29/11/2015 (v0.11.0-rc1) 16 contributors 128 github stars release frequence = 1 every two months	**Starting date** : December 2013 **1st release** : 30/07/2015 122 contributors on geth 3.8k github stars on geth release frequence > 1 per month
based-technology	**Javascript** **IPFS**	**Python** **RethinkDB** or **MongoDB**	**C#** (.NET) **MSSQL** or **Sqlite** or **MongoDB** (possibly others)	**Go lang** (geth) Swarm is developed along with geth	**Go lang** (go-ipfs) **Javascript** (js-ipfs) **Python** (py-ipfs)	**Node.js** (+ Python)	**Go lang**	**Go lang**	**Go lang** (geth), **Rust lang** (parity), **C++** (eth), **Java** (ethereum)), **Python** (pyethapp), **Javascript** (ethereumjs-lib), **Haskell** (ethereumh), **Ruby** (ruby-ethereum) **geth v1.6.1** (113 releases) last commit : 09/05/2017, **parity 1.6.6** (62 releases) last commit : 09/05/2017, **eth 1.3.0** (228 releases) last commit: 09/05/2017, **ethereumJ 1.5.0** (32 releases), last commit : 26/04/2017
current release	**v0.16.0** (17/01/2017)	**v0.10.1** (19/04/2017) **next version v1.0** : early June 2017	**v0.7.1** (07/12/2016)	**POC 0.2 RC5** (geth v1.5) **next version POC 0.4** : Q2 2017	**v0.4.9-rc2** (07/05/2017)	**v6.4.2** (26/04/2017)	**v0.4.2.0** (08/05/2017)	**v0.16.0** (08/04/2017)	
code availability	https://github.com/orbitdb	https://github.com/bigchaindb	https://github.com/openchain	https://github.com/ethereum	https://github.com/ipfs	https://github.com/Storj	https://github.com/FactomProject	https://github.com/monax	https://github.com/ethereum
deployment	npm package	manually / Terraform / Kubernetes / Docker	manually / docker	manually from source using *go install* command	ipfs-update (Go installer for ipfs) manually from a Prebuilt Package (.tar.gz or .exe)	npm package	X	docker (docker-machine) / Debian Package / RPM Package / Binary	**geth** : manually from source using *go install* command / macOS via Homebrew / Ubuntu via PPAs / Windows via **Clients :** **Javascript API** (web3.js), **Java library** (web3j) **.NET library** (Nethereum) **Ruby JSON-RPC wrapper** (ethereum-ruby) **Browsers (+wallet) :** **Mist browser** **Metamask extension** **Development :** **Solidity** (smart contracts language)
additional tools, driver, API...	X	**HTTP API** **Python driver**	**HTTP API** **NodeJS client**	X	**HTTP API**	**Javascript library** (storj.js) **REST API**	**Go lang library** (factom) **JSON-RPC API**	X	

FIGURE E.1 – Comparaison des technologies - Partie 1

10/05/2017	OrbitDB	BigchainDB	Openchain	Swarm	IPFS	Storj	Factom	Monax	Ethereum
mining	No	No	No	Not directly but Ethereum implies mining On the other hand, there are incentives to compensate for bandwidth and storage services with Ether	No	Yes The data is stored by users that are willing to provide their extra storage space (although they cannot access it) in exchange for a monetary reward	not directly *(The consensus algorithm rotates the responsibility for managing a chain over the Federated Servers. So for a minute, one server is in control of your chain The next minute, another server is in control and so forth)*	No	Yes
cryptocurrency	None X	None (but assets instead) Openchain is not really a blockchain but more precisely a transaction chain.	None There are no transactions	Ether There are no transactions	None There are no transactions	SCJX The Storj system enables users to store data in a secure and decentralized manner. It does this by using such blockchain features as a transaction ledger, public/private key encryption, and cryptographic hash functions.	Factoids (FCT) The consensus algorithm rotates the responsibility for managing a chain over the Federated Servers. So for a minute, one server is in control of your chain The next minute, another server is in control and so forth Transactions are	X	Ether a transaction refers to the signed data package that stores a message to be sent from an "user" account to another (user or contract) account on the blockchain and has a cost (depending on gas price and code complexity) which results from miner fees
transactions layer	None Transactions are used to register, issue, create or transfer assets. CREATE transaction can be used to register, issue, create an asset. TRANSFER transaction can transfer an asset (https://docs.bigchaindb.com/en/latest/transaction-concepts.html)	Transactions are used to issue amount of assets	Ability to have multiple Openchain instances replicating from each other using transaction stream which is a websocket endpoint	There are no transactions Swarm is based on 'upload and disappear' philosophy which allows users to upload stuff and potentially disappear, then retrieve content later	There are no transactions IPFS is a distributed file system that seeks to connect all computing devices with the same system of files.	files are both secure and not easily viewed or hacked by unauthorized users which means To protect against node failures and downtime the Storj network defaults to three redundant storage nodes. If your files are stored with multiple nodes and one goes offline, the Storj network will automatically find another node to take over the open contract so your files continue to be available.			
replication	X *bigchaindb add-replicas your-mongod-hostname:27017* native Mongo replication (*rs.add()*) native Rethink replication (set shards) Replica must be added when a new node is being to be connected to the network	Swarm implements a distributed content-addressed chunk store. Chunks are arbitrary data blobs with a fixed maximum size and are represented by a hash which is the address that clients can use to retrieve the chunk If we upload a chunk to the Swarm, the protocol determines that it will eventually end up being stored at nodes that are closest to the chunk's address	Distributed Hash Tables (DHTs) are widely used to coordinate and maintain metadata about peer-to-peer systems. IPFS is peer-to-peer; no nodes are privileged. IPFS nodes store IPFS objects in local storage. Nodes connect to each other and transfer objects. These objects represent files and other data structures.	X					
permissioning policy	X **permissioned** **unpermissioned** **consortium** readable using a public HTTP API	**unpermissioned** : Fully open ledger that can be joined anonymously **permissioned** : Closed-loop ledger where participants must be approved by the administrator. A mix of the above where approved users enjoy more rights than anonymous users. readable using a public HTTP API	**unpermissioned** (public network) **permissioned, consortium** (private network)	**unpermissioned** (public network) **permissioned, consortium** (private network should be possible using ipfs-cluster)	**unpermissioned** (public network)	**unpermissioned** (public network)	**unpermissioned** (public network)	**permissioned** (create your own blockchain)	**unpermissioned** (public network) but permissioning can be added in smart contracts (i.e. using "modifier" keyword which is used as a condition) **permissioned** (private network) **consortium** (http://www.ethdocs.org/en/latest/network/connecting-to-the-network.html#public-private-and-consortium-blockchains)
BFT compliant	X can be turned on but cause a severe dropoff in performance (distributed databases in production today such as DynamoDB, Bigtable, MongoDB, Cassandra, Elasticsearch, are not	No	X	X	X	X	X	X	Proof-of-stake should be BFT I see https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ#how-does-proof-of-stake-fit-into-traditional-byzantine-fault-

FIGURE E.2 – Comparaison des technologies - Partie 2

10/05/2017	OrbitDB	BigchainDB	Openchain	Swarm	IPFS	Storj	Factom	Monax	Ethereum
network topology	distributed Peer-to-peer	Decentralized control via a federation of voting nodes makes for a super-peer P2P network If sharding is turned on (i.e. if the number of shards is larger than one), then the actual data is decentralized in that no one node stores all the data. Every node has its own locally-stored list of the public keys of other consortium members: the so-called keyring. There's no all other BigchainDB nodes in the cluster must add the new node's public key to their BigchainDB keyring. Currently, the only way to get BigchainDB Server to "notice" a changed keyring is to shut it down and start it back up again (with the new keyring).	decentralized	distributed / decentralized Peer-to-peer	distributed / decentralized Peer-to-peer	distributed / decentralized Peer-to-peer	distributed / decentralized Peer-to-peer	distributed / decentralized	The basis for **decentralised consensus** is the **peer-to-peer (distributed) network** of participating nodes which maintain and secure the blockchain Geth finds peers through something called the *discovery protocol*. In the discovery protocol, nodes are gossipping with each other to find out about other nodes on the network
	X	X	Multi-signature is supported. Permissions are expressed using a list of public keys, and a number of require signatures.	X	X	X	X	X	Solidity as language for smart contracts
extras									Ethereum is not designed to store larges files or pieces of data because of the high cost of storage.

FIGURE E.3 – Comparaison des technologies - Partie 3

Résumé

La technologie blockchain s'est popularisée ces dernières années et devrait être largement utilisée dans le futur proche. Malgré l'attention considérable et médiatique, seulement quelques produits exploitent tout le potentiel de cette technologie (e.g., Bitcoin and Ethereum). Pourtant, les propriétés innovantes qu'apporte la blockchain en font un outil idéal pour concevoir des architectures de sécurité.
La Commission européenne a lancé un projet, nommé FutureTrust, pour aborder les mesures présentées dans son règlement sur l'identification électronique (eID) et les services de confiance pour les transactions électroniques sécurisées au sein de l'UE (eIDAS). Le consortium FutureTrust se compose de seize partenaires, dont ARHS Spikeseed, qui sont engagés dans la conception et la mise en œuvre de solutions pour faire appliquer le règlement eIDAS.
Le présent document détaille l'implémentation d'un service de liste de confiance globale basé sur la blockchain. Plus particulièrement, il présente la révision de l'architecture actuelle des listes de confiance, décrit l'implémentation de cette nouvelle architecture, et expose les avantages de l'utilisation d'une blockchain dans le contexte de la confiance numérique. Le service développé a pour rôle de gérer des listes de confiance conforme au standard ETSI 119 612 et de permettre à toute partie intéressée de déterminer si un service de confiance respecte les exigences du standard. D'un point de vue technique, le système mis en place a pour vocation de stocker des données relatives à la confiance numérique dans un registre publique immuable. Le système utilise un réseau pair-à-pair décentralisé et distribué, assurant l'intégrité et la disponibilité des informations ainsi qu'une résilience forte contre les attaques par déni de service.

Mots-clés : eIDAS, blockchain, décentralisation, confiance numérique

Abstract

Blockchain technology emerged over the last few years and should be widely deployed in the near future. Despite the considerable amount of interest and media attention, only a few products exploit the full potential of this technology (e.g., Bitcoin and Ethereum). However, the innovative properties of blockchain make it an ideal tool for devising novel security architectures.
The European Commission initiated a project, denominated FutureTrust, to address the measures introduced in its regulation on electronic identification (eID) and trusted services for electronic transactions in the internal market (eIDAS). FutureTrust's consortium consists of sixteen partners, including ARHS Spikeseed, which are engaged in the design and implementation of solutions to enforce the eIDAS regulation.
The present document specifies the requirements for a blockchain-based global trusted lists management service. More specifically, it introduces the redesign of the current architecture for trusted lists, describes the implementation of this novel architecture, and discusses the advantages of the use of blockchain in the context of digital trust. The developed service aims at managing trusted lists in compliance with ETSI 119 612 and enables any interested party to determine whether a trust service is or was operating in compliance with the relevant requirements. Technically, the system we implemented is used for storing data related to digital trust in an immutable public ledger. The system uses a peer-to-peer decentralized and distributed network, guarantees integrity and availability of information, and improves resilience against denial of service attacks.

Keywords : eIDAS, blockchain, decentralization, digital trust