Conception d'une solution de connexion à partir de terminaux mobiles de type smartphone

Éric Kaminski

CONSERVATOIRE NATIONAL DES ARTS ET METIERS
CENTRE REGIONAL DE LORRAINE

CONCEPTION D'UNE SOLUTION DE CONNEXION A PARTIR DE TERMINAUX MOBILES DE TYPE SMARTPHONE

MEMOIRE EN INFORMATIQUE

OPTION RESEAUX, SYSTEMES ET MULTIMEDIA (IRSM)

Présenté par

Eric KAMINSKI

Soutenu le 14 décembre 2012

JURY

Président :	J.P. ARNAUD	Professeur de Chaire au CNAM, et président du jury
Membres :	S. VIALLE	Responsable Régional de Spécialité informatique au CNAM de Lorraine
	M. IANOTTO	Enseignant-chercheur à Supélec, et tuteur du mémoire
	A. DETANTE	Consultant Senior chez InTech S.A., et responsable de stage
	A. PUCAR	Administrateur associé chez InTech Group

Remerciements

Je tiens tout d'abord à remercier François Bocci, Alain Pucar et Michel Sanitas, directeurs d'InTech Group ainsi que Nathalie Milair, Fabrice Croiseaux et Sébastien Larose, directeurs opérationnels d'InTech S.A., pour m'avoir permis de réaliser mon projet au sein de leur entreprise dans laquelle je travaille depuis le mois de novembre 2005.

Je tiens plus particulièrement à remercier Antoine Detante et Michel Ianotto, mes tuteurs de stage, ainsi que Stéphane Vialle, responsable régional, pour leurs conseils avisés et le suivi de mon travail tout au long de mon projet.

Je souhaite également remercier Guy Muller et Nicolas Vilmart ainsi que l'ensemble des équipes IT de LuxTrust et Mobey que j'ai côtoyé, pour leur soutien dans mon travail.

Grand merci également à Bernard Antoine, Antoine Thiriez et Alexandra Lacroix des équipes opérationnelles de LuxTrust et Mobey.

Enfin, je voudrais féliciter Ludovic Iggiotti pour le travail fourni dans le cadre de son stage de fin d'étude que j'ai eu le plaisir d'encadrer.

Sommaire

1. Introduction

Ce mémoire marque l'achèvement de mon parcours au CNAM de Lorraine où je suis inscrit depuis le mois de septembre 2005. Je souhaitais compléter mon cursus universitaire qui s'est achevé avec l'obtention d'une Licence Professionnelle des Métiers de l'Informatique option Génie Logiciel en 2004. C'est pourquoi je me suis tourné vers la formation Ingénieur en Informatique option Réseaux, Systèmes et Multimédia.

A mon sens, l'ingénieur est d'abord un chercheur. Son rôle va être d'apporter une solution à un problème complexe. Il partira d'un cahier des charges qu'il aura défini avec les utilisateurs afin d'obtenir le résultat satisfaisant. Il est impliqué dans les diverses phases du projet que sont la compréhension des besoins, la conception, la réalisation, le déploiement et la maintenance.

Je me suis donc attelé à trouver un sujet qui me permettrait de couvrir toutes les étapes du cycle de développement d'un projet, de l'analyse de marché à la conception d'une solution pour aboutir au déploiement d'une application.

La première partie de ce mémoire va présenter le contexte du stage et les acteurs impliqués, le stage étant effectué chez InTech S.A. en partenariat avec la société LuxTrust. Dans une seconde partie, je vais détailler un cahier des charges et analyser le marché afin de justifier la solution que je propose et la plus-value que cette dernière va apporter. Dans une troisième partie, je vais réaliser un état de l'art des différentes technologies mobiles ainsi que des algorithmes de chiffrement puis je détaillerai les étapes de la conception jusqu'au déploiement.

2. Le contexte

2.1. Description du projet

Le projet a été planifié dans le cadre de ma formation Ingénieur CNAM. Il a consisté à concevoir et à développer une solution de connexion à un site internet à l'aide d'un Smartphone. La société LuxTrust S.A. a été impliquée dans ce projet pour apporter un volet authentification forte à la solution de connexion. La solution a ensuite été intégrée à la solution FlashIz vendue par la société Mobey S.A.

2.2. LuxTrust S.A.

2.2.1. La société

LuxTrust S.A est une autorité de certification constituée le 18 novembre 2005 par le gouvernement luxembourgeois et d'importants acteurs du secteur privé luxembourgeois, notamment du secteur financier. Elle permet de répondre à un besoin de sécurité accrue dans le commerce électronique, tant pour le gouvernement que pour le secteur financier, les autres acteurs de l'économie luxembourgeoise et les citoyens. Ceci en conservant une vocation internationale à travers l'adoption de standards reconnus dans le monde entier.

Une infrastructure à clé publique (ICP ou *PKI – Public Key Infrastructure* en anglais) est basée sur une technologie qui permet de lier une clé privée, qui est détenue uniquement par l'utilisateur, à une clé publique, qui est connue par tout le monde. Ces deux clés forment une paire unique et une tierce personne peut vérifier, à l'aide de la clé publique, qu'un message ou une signature provient bien d'une clé privée spécifique.
LuxTrust, en tant qu'autorité de certification, rajoute à cette technologie de base des procédures d'un niveau très élevé pour identifier un utilisateur avant de lui donner une clé privée : présentation physique à un guichet, présentation de justificatifs divers, etc.

LuxTrust crée ensuite un certificat électronique qui contient la clé publique de l'utilisateur ainsi que les données relatives à son identité. LuxTrust signe ce certificat

électroniquement afin qu'il ne puisse plus être modifié et pour assurer à un tiers que les procédures d'identification de l'utilisateur ont bien été respectées. Les hauts standards que LuxTrust a retenus pour la technologie utilisée et pour ses procédures font qu'une tierce personne, le fournisseur d'application par exemple, peut faire confiance au contenu du certificat et peut, à tout moment, vérifier sa validité auprès de LuxTrust.

2.2.2. Les produits d'authentification LuxTrust

LuxTrust propose plusieurs produits permettant de s'authentifier. La base de ces produits est le certificat qui contient toutes les informations permettant l'identification. Ce certificat s'intègre dans plusieurs solutions qui diffèrent selon l'usage que l'utilisateur en aura.

2.2.2.1. Fichier .cer

Les fichiers .cer sont des certificats émis au nom d'une personne physique. Ces certificats de type SSL, Wildcard, SAN ou Objet répondent à des standards internationaux et permettent d'identifier des serveurs et des applications en ligne envers des tiers. Il est nécessaire d'avoir une application compatible qui saura intégrer et utiliser le fichier cer.

2.2.2.2. Cartes à puce ou *Smartcard*

Figure 1 - Smartcard

Les Smartcards (figure 1) permettent de s'authentifier en ligne et de signer électroniquement des documents ou des transactions. Les Smartcards sont munies de deux certificats électroniques et de leurs clés privées respectives. Un certificat réservé aux seules signatures électroniques et un certificat réservé à l'authentification. Les Smartcards répondent aux plus hauts standards internationaux ce qui garantit la sécurité technique de la carte. Il est nécessaire d'acheter un lecteur spécifique et d'installer un middleware sur son ordinateur pour utiliser ce type de produit.

2.2.2.3. "*Signing Stick*" USB

Figure 2 - Signing Stick USB

Le Signing Stick LuxTrust (figure 2) se présente sous la forme d'un Stick USB. Il est muni d'une puce qui contient deux certificats électroniques d'une validité de trois ans. Un certificat réservé à la signature électronique et un autre à l'authentification ou à la sécurisation des communications électroniques. Avec ce produit, seul le middleware doit être installé sur l'ordinateur, la totalité des machines étant aujourd'hui équipée de ports USB. Avec ces deux produits, l'utilisateur transporte le certificat avec lui. Il peut également le conserver sur un serveur centralisé dans un environnement sécurisé.

2.2.2.4. Certificats "*Signing Server*"

Les certificats *Signing Server* combinent un haut niveau de sécurité à une portabilité très pratique, permettant d'accéder à vos applications en ligne à tout moment et partout. La clé privée est conservée sur un serveur centralisé dans un environnement hautement sécurisé. Cette solution garantit que l'utilisateur est la seule personne à pouvoir accéder à la clé. Deux solutions sont offertes :

- Le Token (figure 3) est un petit outil (de la taille d'un porte-clés avec un écran LCD intégré) qui génère un code d'accès spécifique et différent toutes les 30 secondes. Combiné à un User ID et à un mot de passe personnel, ce code d'accès permet de s'authentifier dans une application en ligne.

Figure 3 - token LuxTrust

- La solution SMS fonctionne presque de la même façon que celle du Token, mais au lieu d'obtenir le code électronique sur un Token, celui-ci est reçu directement sur un téléphone portable via SMS et ce pour chaque signature et authentification. A l'aide de son User ID et de son mot de passe personnel, l'utilisateur accède en toute simplicité à votre clé privée.

2.3. InTech S.A.

2.3.1. La société

La société InTech S.A. a été créée à Schifflange (Luxembourg) à la fin de l'année 1995. En 2010, elle a réalisé 7,4 M€ de chiffres d'affaires grâce à 74 collaborateurs. InTech est aujourd'hui un acteur reconnu par le marché luxembourgeois et international. La société participe à l'évolution des systèmes d'information de grandes organisations telles que des établissements financiers, des institutions luxembourgeoises et européennes, ou des groupes industriels.

2.3.2. Domaines de compétences

InTech est spécialisé dans la conception et la réalisation de solutions métiers construites à partir de composants industriels fondées sur les Nouvelles Technologies de l'Information. On peut citer les domaines suivants :

- l'administration de fonds : ingénierie des processus, audit interne ou externe, compliance, risk management, base valeurs, pricing de titres, business intelligence.
- la Banque Privée : gestion de la relation Clients, intégration front office/back-office.
- les places financières : gestion de l'information financière, système de workflow optimisant la gestion des flux entre les différents acteurs de l'administration de fonds (Autorités de tutelle, promoteurs, administrations centrales, avocats…).
- les places de marché industrielles : vente de produits en ligne, intégration avec les back-offices.

- les systèmes industriels de gestion de la qualité.

2.3.3. Les offres de services

Les activités d'InTech se répartissent en 5 secteurs distincts comme illustrés sur la figure 4.

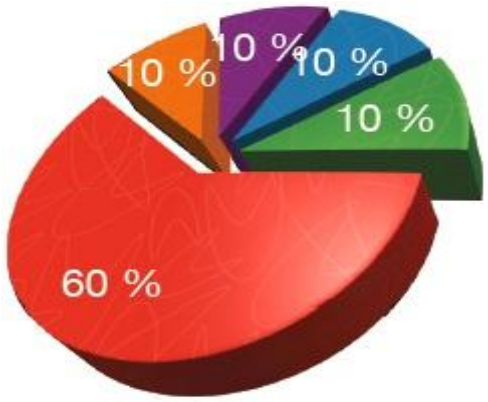

Figure 4 - Répartition des activités d'InTech en pourcentage du chiffre d'affaires

On retrouve les activités suivantes :

- le Conseil (10%) : InTech accompagne ses clients dans la mise en œuvre des nouvelles technologies, lors de la mise en place d'architectures fonctionnelles ou d'architectures techniques. Nous sommes également disponibles pour les assister dans l'organisation de projets ou leur faire profiter de notre expérience en risk management.

- l'Expertise Fonctionnelle (10%) : Notre domaine de compétence couvre également des domaines métiers tels que la gestion des fonds, le fonctionnement des banques privées, des places de marché financières et des places de marché industrielles.

- l'Expertise Technique (10%) : Nous intégrons également des équipes "Architecture" pour apporter notre expérience dans des domaines tels que la conception orientée Objet, la conception et l'implémentation de frameworks orientés Objet et l'implémentation de serveurs d'applications. Nous pouvons former les équipes de nos clients sur les technologies Web/Objet ou encore la conception et l'implémentation de systèmes de workflow.

- l'Intégration de Systèmes (60%): La majorité de notre effectif est toutefois alloué à la constitution d'équipe projet en régie chez nos clients. Ce dernier, maître d'ouvrage, demande à InTech de prendre en charge, la maîtrise d'œuvre d'un projet, en assumant les fonctions de management, de conception, de réalisation, de transfert d'expérience sur ce projet. Il s'agit là de l'activité principale d'InTech.
- l'Assistance Technique (10%) : InTech met à disposition du client une ressource dont la compétence est prédéfinie. Cette prestation est mise en œuvre pour des clients privilégiés, car elle ne correspond pas au cœur de métier de notre Entreprise.

2.3.4. Organisation

InTech est dirigé par un comité exécutif de trois associés, qui tout en appliquant des pratiques de management éprouvées, privilégie la dimension humaine nécessaire à l'écoute active de ses clients et à la mobilisation de ses collaborateurs. Chaque associé est l'animateur d'une fonction principale de l'Entreprise. François Bocci est responsable des ressources humaines en charge de la gestion prévisionnelle des emplois et des ressources. Il s'attèle à l'étude des rémunérations, du recrutement et de la formation des collaborateurs.

Michel Sanitas assure le management de la technique et des opérations. Ceci comprend la veille technologique, les systèmes qualité, la gestion des offres ainsi que la supervision des projets et la planification des ressources.

Alain Pucar anime la fonction marketing et commerciale. La tâche qui lui incombe englobe la planification stratégique, les études de marché les analyses concurrentielles, et l'élaboration des offres génériques (en collaboration avec la direction technique. Il se charge également de la communication et de la vente aux clients existants et nouveaux.

Les autres collaborateurs assurent des fonctions d'architectes fonctionnels et techniques, d'analystes, de développeurs ainsi que des managers. La figure 5 montre la répartition de ces fonctions au sein de la Société.

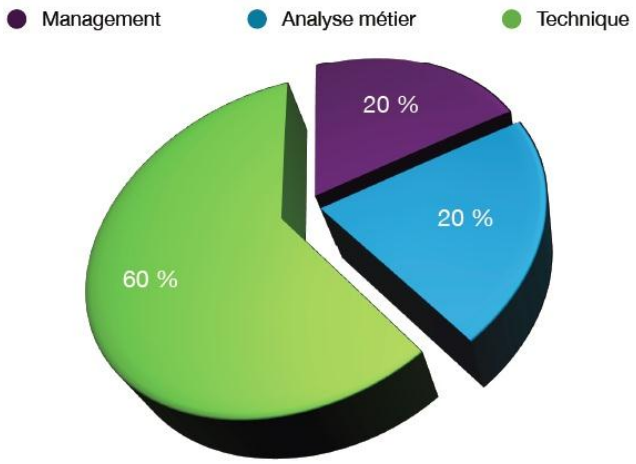

Figure 5 - Répartition des collaborateurs par domaine

2.4. Mobey S.A

2.4.1. La société

Mobey est un établissement de monnaie électronique créé en 2011. Cela signifie qu'à la manière de Paypal, cette société est agrée par la CSSF (Commission de Surveillance du Secteur Financier) pour proposer à ses clients des solutions de paiements électronique. Rassemblant des professionnels du secteur financier, de l'industrie des paiements et des spécialistes en technologie mobile, Mobey a mis au point et distribue aujourd'hui sa solution Flashiz.

2.4.2. Le produit FlashIz

Flashiz est une solution qui permet d'effectuer des paiements au profit d'un particulier ou d'un professionnel (commerçant) : l'utilisateur accède au service de paiement depuis son Smartphone connecté à Internet via une application mobile dédiée qui permet de scanner un QR code. L'utilisateur peut alors valider le paiement, qui est immédiatement déduit de son solde pour être crédité sur le solde du bénéficiaire. On peut ainsi distinguer :

- les paiements des achats chez des commerçants : le paiement se fait directement auprès du commerçant (bars, restaurants, magasins) à l'aide d'un QR code, soit affiché sur une tablette, soit annexé au ticket de caisse ;
- les paiements sur Internet en choisissant Flashiz comme mode de paiement;

- les paiements de particulier à particulier en flashant avec son Smartphone un QR sur l'écran d'une deuxième Smartphone;
- les paiements de factures papier depuis chez soi.

Après avoir créé son compte sur www.flashiz.com et avoir téléchargé l'application mobile Flashiz sur son Smartphone. Le compte Flashiz peut à tout moment être chargé, soit par virement bancaire, soit par cash auprès des commerçants agréés, soit encore par carte de crédit.

3. Cahier des charges

3.1. Description de l'architecture existante

La figure 6 illustre le fonctionnement de la solution actuellement proposée par LuxTrust. Les utilisateurs vont se connecter à un site internet sur leur station de travail (étape 1). La demande se fait à LuxTrust au travers d'un middleware installé sur l'ordinateur (étape 2). Lorsque LuxTrust reçoit les paramètres de connexion entrés par l'utilisateur, ses serveurs valident l'authentification (étape 3) et l'utilisateur peut alors accéder au contenu privé du site internet (étape 4).

Figure 6 - Solution LuxTrust existante

Actuellement, l'utilisateur qui souhaite s'authentifier sur un site va fournir à ce site un login, un mot de passe et un OTP LuxTrust. Il est nécessaire d'installer un plug in sur sa machine afin de pouvoir utiliser les moyens mis à disposition par LuxTrust pour s'authentifier. Ce plugin est difficile à mettre en place et peut rebuter les néophytes. Tout le processus d'utilisation pourrait être repensé en utilisant le potentiel offert par les Smartphone. Les plateformes de téléchargement regroupent les applications compatibles avec le téléphone. Il suffit de cliquer pour que la bonne version soit téléchargée et installée. Si l'application qui communique avec LuxTrust est installée sur le Smartphone et que la connexion à un site internet se fait à partir d'une station de travail, il faut trouver un moyen de faire communiquer les deux appareils. Là encore on peut utiliser les possibilités offertes par les Smartphone car tous les modèles sont équipés d'appareil photo. Les informations transmises par le site client pourraient donc être "lues" par le téléphone qui se chargerait ensuite de valider l'authentification chez LuxTrust. Il ne restera alors qu'à trouver le moyen de forcer le rafraîchissement du site internet dans le navigateur de l'utilisation une fois l'authentification validée.

3.2. Etude des besoins fonctionnels et non fonctionnels

3.2.1. Les besoins fonctionnels

La solution proposée doit permettre d'atteindre les objectifs suivants :

- l'objectif du travail demandé est de rendre possible l'utilisation des certificats avec un appareil mobile de type *Android, iPhone* ou *BlackBerry*, tout en respectant au maximum les standards de sécurité actuellement utilisés afin de garantir une connexion véritablement sécurisée.
- le Smartphone devient alors un moyen d'authentification forte pour se connecter à un serveur sécurisé ou échanger des fichiers chiffrés sans qu'aucune donnée ne transite en clair entre ce dernier et l'appareil mobile.
- la façon de se connecter à des sites internet de façon devient plus conviviale grâce à l'application mobile est capable de s'authentifier sur un serveur d'authentification. Ce dernier renvoie un signal au serveur Web afin de rafraichir la page dans le navigateur de l'utilisateur.

Ceci devient possible grâce à l'implémentation de la couche d'abstraction LuxTrust sur les Smartphone. Toute l'architecture respecte des critères de sécurité élevés empêchant des données sensibles de transiter en clair sur le réseau.

3.2.2. Les besoins non fonctionnels

L'objectif de ce projet est de proposer une nouvelle façon de se connecter à des sites internet. Pour permettre aux serveurs web de nos clients d'utiliser les nouvelles fonctionnalités, il sera nécessaire de procéder à quelques adaptations de leur côté. Il y aura donc une librairie à intégrer aux serveurs web et c'est ici que nous rencontrerons une véritable contrainte. En effet, il nous faudra trouver le juste milieu entre une solution assez évoluée techniquement pour atteindre l'objectif que nous nous sommes fixé dans le cahier des charges, mais il sera nécessaire de s'assurer que l'intégration se fasse de façon suffisamment simple pour ne pas rebuter nos partenaires. L'intégration aux serveurs de LuxTrust devra également pouvoir se faire sans révolutionner complétement l'architecture mise en place. Il faudra discuter avec les équipes en place de la meilleure façon de leur amener cette intégration, à travers un module complémentaire ou une librairie à intégrer à l'application existante.

Enfin le cœur du projet va être le portage du Common Layer sur une plateforme de type Android ou iOS. Le développement devra être réalisé clairement afin de pouvoir être proposé ultérieurement à des développeurs sous forme de librairie intégrable à leurs projets personnalisés. La dernière contrainte va concerner le déploiement de l'application. Cette étape se réalise via les plateformes de distribution Google Play pour Android et Apple Store pour Apple. Il existe toutefois des standards à respecter pour être autoriser à diffuser une application. Il faudra également respecter ces standards pour permettre aux utilisateurs de télécharger et d'installer simplement notre application. Les fonctionnalités attendues sont reprises dans le diagramme des cas d'utilisation ci-dessous (figure 7) :

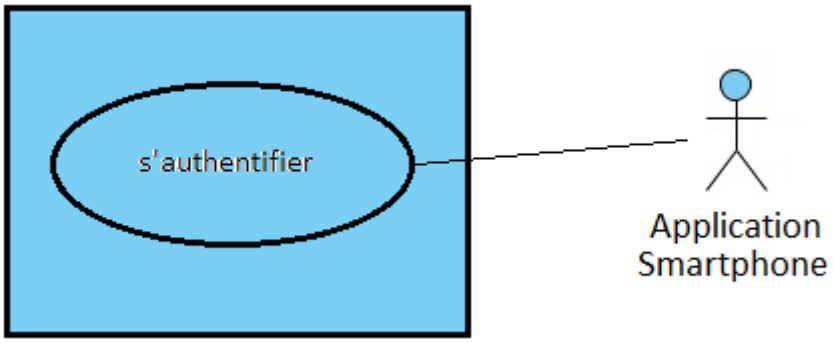

Figure 7 - Diagramme de cas d'utilisation

Les actions réalisées par chaque acteur du système seront détaillées plus loin dans ce mémoire. Nous reviendrons sur le protocole de connexion implémenté dans ma solution.

3.3. Analyse de marché

A mon sens, un véritable stage de niveau ingénieur doit déboucher sur un projet qui, même s'il présente un intérêt technique, doit être motivé par des éléments de marché pertinents. Une analyse du développement des Smartphones sur le Luxembourg et dans les pays de l'Europe de l'Ouest a donc été réalisée grâce à des informations collectées par des cabinets comme ComScore, GfK, Deloitte et IP Luxembourg.

En 2011, 56% des Luxembourgeois de plus de 18 ans sont possesseurs d'un Smartphone. Ce nombre s'élève à 39% pour la France, le Royaume-Unis, l'Allemagne, l'Italie et l'Espagne. Pour ces 5 pays, le pourcentage était de 19% en 2009 et de 27% en 2010. En se basant sur ces valeurs, ComScore estime une croissance de 10% par an. Cet engouement s'explique par l'évolution de la technologie et des prix d'achat qui sont inversement proportionnels. La technologie évolue rapidement et nous permet d'acheter

des téléphones ayant des possibilités très vastes et tout un champ de fonctionnalités. Parallèlement, le prix de ces téléphones baisse et ils ont tendance à se démocratiser. Selon IPLux, 91% des possesseurs de Smartphones utilisent leurs appareils pour surfer sur Internet ou utiliser des applications. Ce pourcentage devrait baisser, IP Lux l'estime à 85% en 2015, mais ce sera dû au fait que les Smartphones tendent à devenir la norme des téléphones et même un utilisateur non intéressé par les possibilités d'un tel appareil se verrait tout de même contraint d'en acheter un.

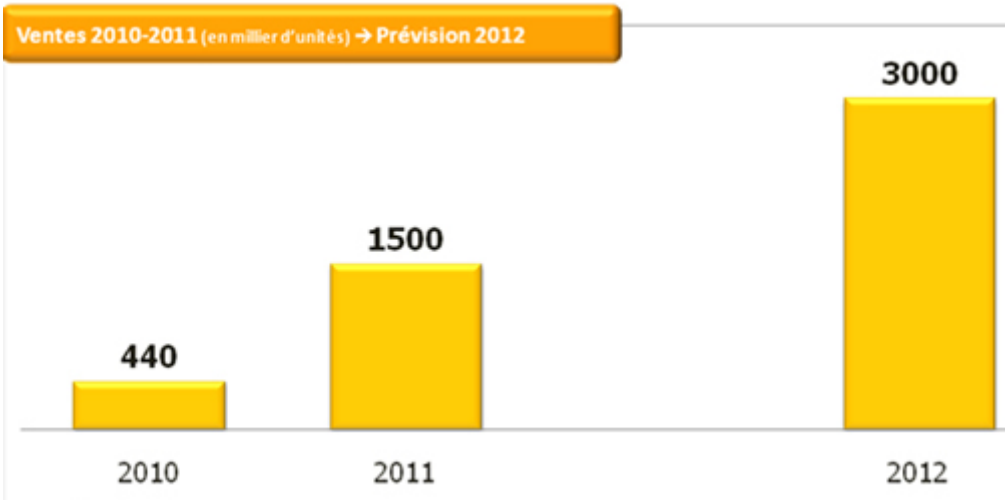

Figure 8 - Prévision sur les ventes de tablettes

Depuis deux ou trois ans le marché de la mobilité explose. Cela s'illustre aussi avec la vente de tablettes. Selon le cabinet GfK, 500 000 tablettes se sont vendues en France en 2010, le nombre est passé à 1,5 millions d'unité en 2011. Le cabinet Deloitte estime qu'un million de tablette iPad 2 se sont vendus à sa sortie en février 2012 et les prévisions font état de 3 millions de tablettes vendues (figure 8).

On constate donc que le marché de la mobilité est en plein essors et que les techniques de marketing doivent évoluer pour prendre en compte ce nouveau support. Certaines prévisions laissent même envisager que la navigation internet sur mobile (Smartphone ou tablette) pourrait dépasser la navigation internet depuis un ordinateur (figure 9) en 2013 :

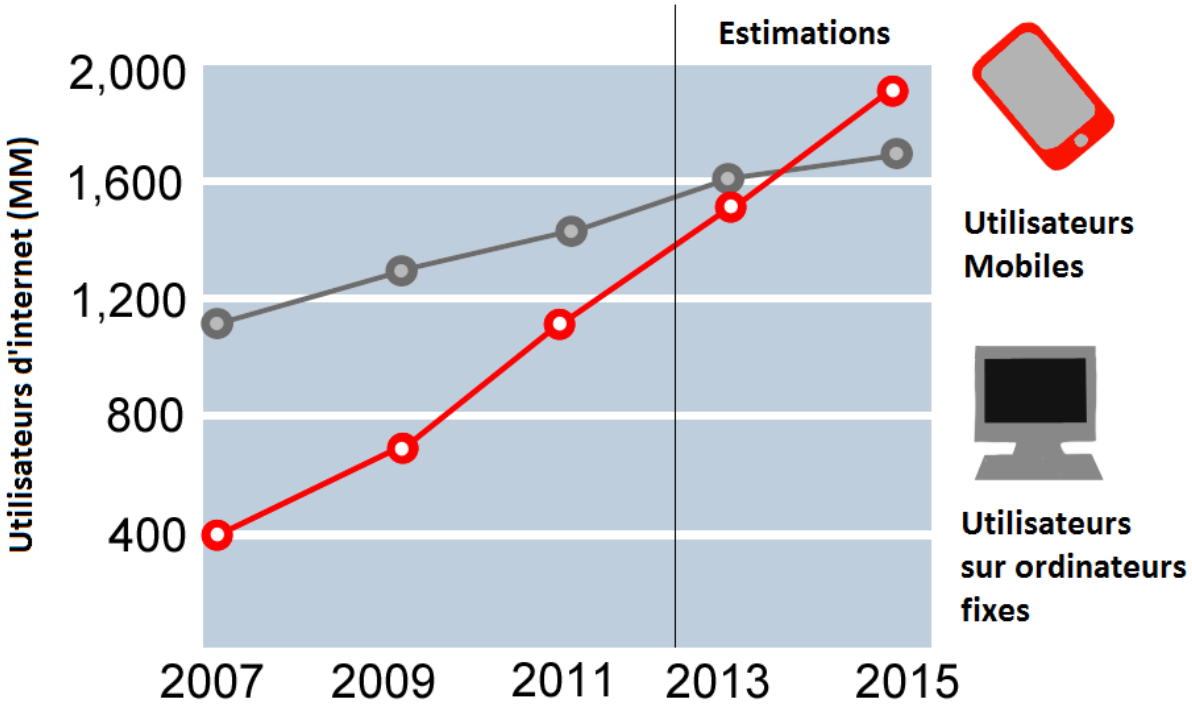

Figure 9 – Comparatif sur le nombre d'utilisateur d'internet par plateforme

Fort de ce constat, la question de la navigation sur des sites sécurisés se pose. De nos jours, les banques proposent des solutions de Webbanking. Si ces applications sont de plus en plus utilisées à partir de téléphones portables, il va falloir se poser la question de la sécurité. Sur le marché Luxembourgeois, l'authentification forte est le domaine de LuxTrust S.A. qui est assermenté pour créer, distribuer et gérer des certificats. Or, à ce jour, il n'existe pas de solution LuxTrust adaptée à la navigation nomade. Alors que les utilisateurs se connectent de plus en plus et partagent de plus en plus de choses sur internet, il se développe un véritable besoin de sécurité.

3.4. Les acteurs implantés sur le marché

Le principal axe du projet est l'authentification forte sur un appareil de type Smartphone, un partenariat avec la société LuxTrust qui s'est imposé comme acteur majeur dans ce domaine sur la place Luxembourgeoise était naturel. Ces derniers ont été convaincus suite à la présentation argumentée de l'analyse de marché réalisée dans le cadre du projet. Convaincu par l'opportunité que représente la solution, ils ont accepté de

collaborer à la conception d'un prototype pour prouver la faisabilité du projet (un Proof of Concept) ceci afin d'ouvrir des négociations avec Cryptomatic.

De là est parti le projet de porter le Common Layer sur un appareil de type Smartphone. La solution de connexion par scan de QR Code n'intéressait pas LuxTrust. Or nous voulions pouvoir l'implémenter dans un environnement réel. InTech s'est associé à la société Mobey SA pour prendre en charge leurs développements informatiques. Le produit de cette start-up (FlashIz) permet d'effectuer des paiements à l'aide son mobile en scannant simplement le QR Code d'une facture. Mobey s'est montré intéressé pour intégrer la solution à leur architecture afin de faire évoluer leur site internet (http://www.flashiz.com). Mobey est intéressé par deux choses en particulier. Tout d'abord, l'opportunité d'intégrer notre solution de connexion à leur site internet est une façon de garder une certaine cohérence dans le service qu'ils proposent. Alors que tous les transferts d'argents sont réalisés via l'application mobile en scannant des QR Code, il est dommage de devoir saisir des identifiants manuellement pour pouvoir gérer son compte sur le site www.flashiz.com. L'autre aspect qui intéresse vivement Flashiz est l'implémentation du Common Layer de LuxTrust à leur application mobile. La grande majorité des clients potentiels contactés par les équipes de Flashiz émettent des doutes sur la fiabilité des transactions. Ajouter l'estampille LuxTrust à leur produit apporterait une plus-value qui rassurerait les détracteurs. Comme le cadre du projet n'entre pas dans un champ commercial, il est impossible à ce jour de proposer l'utilisation du Common Layer à Flashiz. Il a toutefois été proposé d'intégrer la solution de connexion à leur site internet. Il s'agira là du dernier prototype réalisé.

3.5. Les solutions possibles pour répondre au cahier des charges

3.5.1. La solution Gateway

Concernant la compatibilité du Common Layer et des solutions mobiles, il faut préciser que LuxTrust avait déjà analysé ce créneau. Du fait qu'il n'est pas compatible avec les technologies mobiles, ils ont choisi de proposer une solution alternative baptisée le Gateway. Le principe du Gateway, illustré par la figure 10, est de permettre à l'utilisateur de s'authentifier sur un site internet depuis son téléphone. Lors d'une demande de connexion, le site va rediriger l'utilisateur vers les serveurs de LuxTrust via une *frame* intégrée à son site. Les identifiants (login, mot de passe et OTP) sont alors saisis sur le

téléphone et transmis en HTTPS avec un simple chiffrement SSL. Lorsque LuxTrust a validé l'authentification, un signal est renvoyé vers le téléphone portable dont le navigateur est redirigé vers les pages sécurisées du site internet.

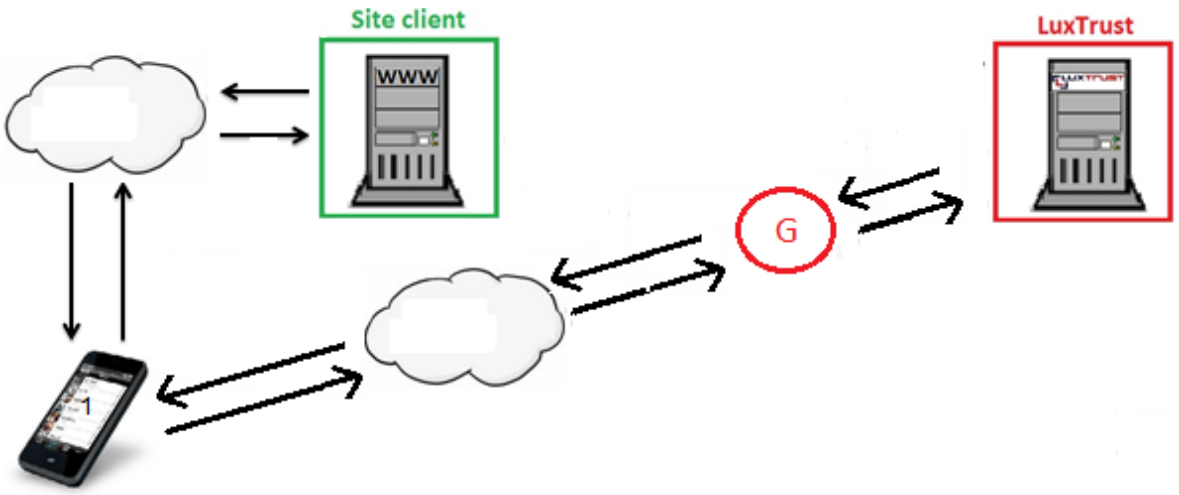

Figure 10 – Architecture de la solution *Gateway*

Le Gateway est physiquement situé chez situé chez LuxTrust et intègre le Common Layer. Il fait la passerelle (*gateway* en anglais) entre le téléphone et les serveurs LuxTrust. La solution Gateway est actuellement proposée à certains sites de *web-banking* Luxembourgeois qui souhaitent mettre en place un degré de sécurité intégrant les standards LuxTrust. Avec un téléphone compatible, l'utilisateur peut se connecter au site pour y transmettre son login, son mot de passe et l'OTP fourni par le token. Or la présence d'un intermédiaire entre l'utilisateur et LuxTrust est un défaut qui dégrade nettement le degré de sécurité. De plus, le fait que les paramètres personnels de l'utilisateur transitent en HTTPS n'est pas non plus un gage de confiance optimal. LuxTrust le reconnait et associe la solution Gateway à un degré de confiance moindre par rapport à une solution sur un ordinateur qui intègrerait directement le Common Layer.

Si cette solution aurait pu être éventuellement utilisée pendant un temps c'est parce que lors de son lancement, le réseau GPRS semblait inviolable, ou du moins n'était pas encore la cible des pirates. Depuis quelques années des statistiques sont publiées et indiquent que les Smartphones sont victimes de leur succès. Les failles commencent à être mises en valeur et les attaques se multiplient. On voit même des antivirus apparaitre sur les plateformes Google Play et Apple Store. La solution ne serait donc pas

une alternative au Gateway mais une évolution nécessaire pour la sécurité sur les connexions nomades.

3.5.2. La solution retenue

Le véritable frein à l'utilisation de la solution Gateway est le faible degré de confiance lors de la transmission de données entre le téléphone et les serveurs de LuxTrust. La solution retenue est donc l'implémentation du Common Layer sur les Smartphone de type Android et iPhone. De cette façon, les données qui transitent sont chiffrées selon un protocole sécurisé et propriétaire. Une application Smartphone permettra à l'utilisateur de s'authentifier sur les serveurs LuxTrust et de se connecter ensuite à un site internet comme illustré par la figure 11.

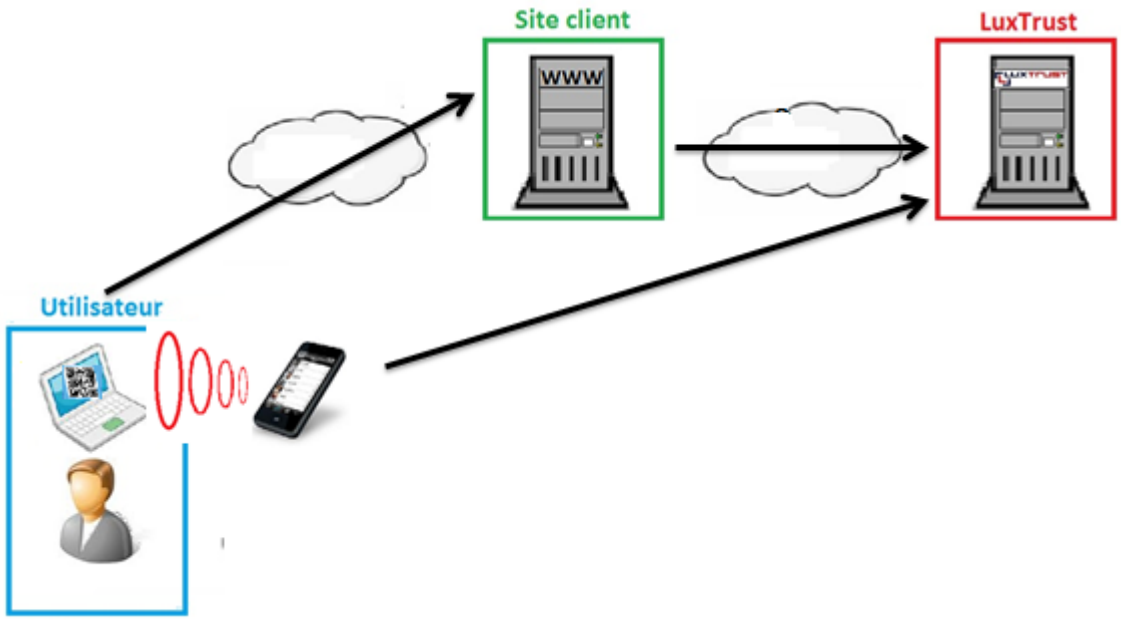

Figure 11 - Représentation de la solution

Les communications entre le site client et les serveurs de LuxTrust seront feront par des appels à des web services.

3.5.3. Principes de l'application et avantages

La première avancée qu'offre cette solution est la possibilité de se connecter à un site

internet en utilisant son Smartphone. C'est en effet un appareil que la majorité des gens utilisent très souvent dans une journée, il est donc plus difficile de le perdre. Si le téléphone n'est plus à sa place on se rend rapidement compte de son absence. L'implémentation du Common Layer sur le téléphone permet une authentification forte directement sur l'appareil car le protocole implémenté est propriétaire et sécurisé. La sécurité est d'ailleurs un des aspects amélioré par la solution du fait que l'authentification se fait maintenant entre trois membres que son l'utilisateur avec son Smartphone, le serveur web qui gère le site internet et le serveur d'authentification basé chez LuxTrust :

- le site demande un jeton à Luxtrust.
- Luxtrust renvoi un jeton, une chaine de caractère unique.
- le site affiche le jeton sous forme d'un QR Code et se met en attente.
- l'utilisateur scanne le jeton sur le site et certifie à Luxtrust qu'il valide l'authentification via son application mobile.
- Luxtrust valide au site qu'il a été contacté pour confirmation par l'utilisateur.
- l'accès est validé.

Car aujourd'hui avec le Gateway :

- l'utilisateur effectue une demande de connexion sur un site.
- Le site redirige l'utilisateur sur une page du site de LuxTrust
- l'utilisateur saisit ses identifiants sur le site sans chiffrement particulier.
- Luxtrust valide l'exactitude des paramètres saisis et redirige l'utilisateur sur le site d'origine.

L'utilisation du Common Layer va permettre notre application de chiffrer les données qui vont transiter entre LuxTrust et l'application mobile. De plus, la solution Gateway ne permet que l'authentification alors que l'implémentation du Common Layer nous ouvre les portes d'une multitude de possibilités. L'utilisateur pourra récupérer son certificat afin de chiffrer ou déchiffrer des données depuis son téléphone. Il sera également possible de signer des documents ou des e-mails. Enfin, un autre avantage de la solution est la centralisation de l'authentification. Le système s'apparente un peu aux principes du SSO (*Single Sign-On*). Il s'agit d'une méthode permettant à un utilisateur de ne procéder qu'à une seule authentification pour accéder à plusieurs applications informatiques (ou sites web sécurisés). Les objectifs sont multiples :

- simplifier pour l'utilisateur la gestion de ses mots de passe : plus l'utilisateur doit gérer de mots de passe, plus il aura tendance à utiliser des mots de passe

similaires ou simples à mémoriser, abaissant par la même occasion le niveau de sécurité que ces mots de passe offrent ;

- simplifier la gestion des données personnelles détenues par les différents services en ligne, en les coordonnant par des mécanismes de type méta-annuaire ;
- simplifier la définition et la mise en œuvre de politiques de sécurité.

3.6. Gestion prévisionnelle du projet

Lorsque le sujet de ce projet a été rédigé au mois d'octobre 2011 pour être validé, le planning prévisionnel représenté par la figure 12 a été établi.

Figure 12 - Chronologie prévisionnelle du projet

L'étude se déroulera chez InTech S.A. au 17-19 avenue de la Libération à Schifflange (Luxembourg) sous la responsabilité d'Antoine Detante, Consultant Expert. Je serai affecté seul à toutes les phases du projet, de l'analyse à la mise en recette. Les différentes étapes planifiées sont reprises dans la figure 13.

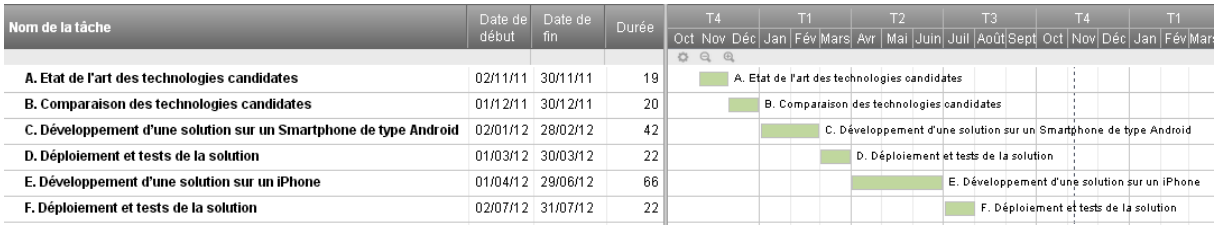

Figure 13 - Ordonnancement prévisionnel des phases du projet

Le deux premiers mois du projet seront consacrés à l'élaboration d'un état de l'art des technologies existantes et des solutions de chiffrement. Je m'appliquerai à recenser les différents systèmes d'exploitation utilisés sur les périphériques mobiles de type Smartphone. Je m'intéresserai aux différents langages de programmation, leurs avantages, leurs contraintes mais également leurs communautés de développeurs ou d'utilisateurs qui se sont formées. L'analyse de toutes ces informations va permettre de valoriser le choix des

plateformes cibles pour ce projet. Dans l'optique de comprendre le fonctionnement du Common Layer de LuxTrust, il sera utile de s'intéresser également aux algorithmes de chiffrement les plus courants et aux méthodes de hashage. Lorsqu'une base de connaissance aura été acquise, il sera possible de commencer la partie programmation avec le développement d'une solution sur une plateforme de type Android. Cette application sera développée en Java et devrait durer deux mois. Nous avons estimé à un mois la période nécessaire pour tester l'application dans un environnement réel. Lorsque la solution sera opérationnelle et stable, je débuterai le portage sur un Smartphone de type iPhone. La durée de cette tâche a été évaluée à trois mois du fait de la difficulté relative à l'absence de Java sur cette plateforme. Le Common Layer étant écrit en Java, il faudra donc en analyser le fonctionnement pour réécrire ses méthodes avec une autre technologie. Le temps restant sera consacré aux tests de la solution iPhone dans un environnement réel. Au terme de ces neuf mois, je m'attèlerai à la rédaction de mon mémoire Ingénieur CNAM.

.

4. Conception et réalisation

4.1. Etat de l'art

4.1.1. Les environnements de développement des Smartphone

4.1.1.1. BlackBerry OS par RIM[1]

La grande force des BlackBerry (figure 14) a longtemps résidé dans leurs capacités à rédiger aisément les mails échangés entre leurs clients, souvent professionnels. Cela est notamment dû aux algorithmes et services de synchronisation mis en place afin de compresser la taille des mails et de leurs pièces jointes d'une part, et d'autre part afin de diminuer la latence d'envoi et de réception des mails d'un client à l'autre. Ayant donc longtemps été un rival de Windows mobile et de son service ActiveSync, il est aujourd'hui complètement compatible avec les services de synchronisation rendus par Microsoft Exchange pour :

Figure 14 - Smartphone Blackbery

- les comptes mails,
- les calendriers,
- les notes,
- les tâches

Les développeurs peuvent concevoir des applications personnelles et partageables à l'aide des API développées soit pour Microsoft Exchange soit pour et par RIM. La marque Canadienne envisagerait d'équiper ses Smartphone avec un nouvel OS, Qxn qui serait compatible avec technologies du HTML5 mais également avec Flash, Adobe AIR, OpenGL ainsi que Java. Les applications BlackBerry sont développées en Java Micro Edition (J2ME). Les principales API utilisées sont incluses dans le BlackBerry Java SDK, le BlackBerry Messenger Social Plateform et le BlackBerry Widget SDK.

[1] Research In Motion

Le premier SDK regroupe les librairies les plus utilisées dans le développement d'application pour BlackBerry OS. Il se compose principalement des API suivantes :

- UiApplication : Cette API regroupe des outils pour gérer le cycle de vie des applications et des interfaces utilisateur.
- InvokeApplication : Elle permet d'exploiter les fonctions à disposition sur le téléphone. On parle d'API d'intégration d'applications. Il devient alors possible de faire appel à la liste des Contacts, à l'Agenda ou encore à la Calculatrice.
- MessageListener qui remplace MessageConnection : Cette API nous donne la possibilité d'accéder à la gestion des SMS et des MMS du téléphone. Des outils permettent également l'émission de mini messages.
- Net.rim.blackberry.api.mail et net.rim.blackberry.api.mail.event : Ces deux API offrent des outils permettant d'exploiter la messagerie et les serveurs de mail BlackBerry.
- Transport : Les méthodes permettant de gérer les transmissions d'informations sur un réseau sont rassemblées dans cette API.
- FileConnection, Database, PersistentStore, RuntimeStore, RMS : Ces API offrent les outils permettant l'accès à la mémoire du téléphone. Le stockage de données devient possible au travers d'une application.
- RIM Cryptographic API : Il s'agit là de l'API cryptographique spécifique à RIM. Elle intègre les différents outils qui implémentent les algorithmes de chiffrement nécessaires à la messagerie sécurisée, aux connexions sécurisées, à l'accès aux bases de stockage de clés et à la gestion des certificats.

Le second SDK, BlackBerry Messenger Social Platform SDK, propose des outils pour utiliser l'application BlackBerry Messager. Cette dernière est toutefois de moins en moins utilisée, remplacée progressivement par d'autres applications mobiles multiplateformes qui ouvrent l'accès aux réseaux sociaux.

A partir du système d'exploitation BlackBerry OS 5.0, il est possible d'utiliser le SDK BlackBerry Widget. Ce dernier met à la disposition des développeurs des outils pour créer des applications Web compatibles BlackBerry. Les applications Web ont l'avantage d'être facilement portable d'une plateforme à une autre et d'offrir des interfaces pour évoluées, plus conviviales. Il n'est malheureusement pas possible d'accéder aux possibilités existantes sur le téléphone comme l'appareil photo, l'agenda ou la messagerie.

Les contraintes de développement

Actuellement, le navigateur du BlackBerry est en effet destiné à une utilisation bas-débit, et impose donc certaines contraintes [Allen S., Graupera V. et Lundrigan L., 2010]. Les champs sont automatiquement redimensionnés et alignés les uns au-dessous des autres, au bord de la page afin de permettre une navigation verticale adaptée à ce type de terminal. Le système d'exploitation est stable et robuste mais très sécurisée. L'utilisation de certaines fonctionnalités (communication SMS ou mail, push, navigation web) ne se fait pas sans contraintes. La dernière version du BlackBerry OS (version 7) intègre un véritable navigateur web mais les parts de marché de RIM se sont écroulées en 2011, supportant difficilement la croissance d'Android et de l'iOS.

Les communautés

RIM vise surtout le marché des professionnels. Il s'agit de personnes qui vont utiliser le téléphone pour récupérer leurs e-mails ou leur agenda en temps réel. La communauté n'est pas aussi active que celle d'Apple. On trouve toutefois un forum qui s'autoproclame "Première communauté française BlackBerry" (http://www.blackberry-france.com/) mais les développeurs francophones restent une denrée rare. Une bande de passionnés a fondé le site CrackBerry.com en 2007. Ce site est devenu une référence aux Etats-Unis pour les utilisateurs et les développeurs.

4.1.1.2. iOS par Apple

Le système équipe les terminaux mobiles et communiquant de la marque : iPhone (figure 15), iPod Touch et iPad. En 2008, il prend le nom d'iPhone OS jusqu'en juin 2010 où il devient iOS. Il est fondé, comme son homologue Mac OS X, sur les noyaux Darwin et XNU. Deux types d'applications cohabitent sur cette plateforme. On trouve par ordre de complexité de développement croissante : les

Figure 15 - iPhone

WebApps puis les applications natives. Les premières sont conçues en HTML et Ajax tandis que les deuxièmes utilisent les différentes couches du système. Deux défauts caractérisent iPhone OS aujourd'hui : l'absence de support du format flash et l'absence de machine virtuelle Java empêchant donc l'utilisation d'applications conçues avec ce langage. Les développements pour iOS sont réalisé dans un langage appelé Objective-C. L'Objective-C est un langage de programmation orienté objet réflexif. C'est une extension du C ANSI, comme le C++, mais qui se distingue de ce dernier par son faible typage, son typage dynamique (le type des variables est contrôlé à l'exécution) et son chargement dynamique (l'Objective-C s'exécute dans un *runtime*, à la manière de Java). Il est une surcouche du langage C, cela signifie donc que n'importe quel code développé en C est compilable avec un compilateur Objective-C. Les deux langages sont compatibles. C'est une autre différence avec le C++. Les spécifications du langage ont récemment été mises à jour avec le passage en version 2.0 qui permet maintenant l'utilisation du ramasse-miette ou Garbage Collector. Il s'agit d'un processus qui s'exécute en parallèle de toute application pour libérer les zones mémoire allouées à des objets qui ne sont plus utilisés. Il n'existe qu'une seule API pour développer sous iOS, elle se nomme Cocoa. Elle se décline toutefois en trois frameworks bien spécifiques :

- *Foundation Kit*, ou plus simplement *Foundation* **:** On l'appelle "*Fondation*" car il regroupe les outils permettant de développer des applications simples avec des composants basics.
- *Application Kit* ou *AppKi* **:** Ce framework regroupe des composants plus évolué et permet de créer des interfaces utilisateurs plus complexes grâce à un grand nombre de composants graphiques
- *Core Data* ou frameworks de persistance **:** Les outils de ce framework vont permettre d'exploiter les composants persistants sur le disque. Il devient alors possible d'exploiter la mémoire du téléphone et son contenu.

Les contraintes pour le développement

Les principales contraintes de développement sont l'absence de moteur flash et surtout l'absence de JVM qui empêche tout développement Java[2]. Pour cette raison Apple a misé sur l'Objective-C [Allen S., Graupera V. et Lundrigan L. 2010]. Il est nécessaire de

[2] Steve Jobs, le fondateur d'Apple, n'était pas convaincu par ces deux technologies

développer en Objective-C dans un environnement Apple, c'est-à-dire sur un ordinateur de type Mac. Une autre contrainte va impacter nettement la distribution d'une future application. En effet, les règles à respecter pour diffuser son projet sur l'AppStore sont très strictes. Il est donc nécessaire de respecter les standards imposés par Apple si on veut qu'une application soit distribuée.

Les communautés de développement

Apple a su se modeler une image décalée. Avec un minimum de communication elle entretient le mystère en créant de très bons produits qui s'adaptent aux besoins des utilisateurs. Les gens deviennent vite fan d'Apple et considèrent leurs appareils comme des objets de mode. Ils ont une certaine fierté à les montrer. Une véritable communauté de gens s'est fédérée autour des produits Apple qui a réussi à rendre ses utilisateurs captifs mais consentants. Lorsqu'on s'équipe avec un objet Apple, il est difficile d'exploiter toutes les possibilités de ce dernier sans être équipé d'un Mac et d'un compte iTune. Avec l'iPad et l'iPhone, Apple ouvre l'AppStore et "autorise" les gens à travailler bénévolement pour eux en développant des milliers d'applications qui seront mises à la disposition des utilisateurs Mac. Il se forme alors une très forte activité sur Internet avec des forums et des blogs d'entraide.

4.1.1.3. Windows Mobile, par Microsoft

Windows mobile n'est pas conçu comme l'iOS. Il peut exploiter de nombreux appareils aux fonctionnalités variées. C'est ce qui le rend aujourd'hui présent sur plusieurs marchés : Smartphone, PDA, système embarqué (donc avec des problématiques liées au calcul temps réel)... La première version de Windows conçu pour des périphériques mobiles est apparue en 2000 sous le nom Pocket PC. Se sont succédées les versions 2002 puis Windows mobile 2003 et 2003 SE et enfin Windows mobile 5, 6, 6.1, et aujourd'hui Windows 7. Les licences sont vendues aux plus gros fabricants mondiaux de smartphones, PDA ou systèmes embarqués, le cinquième étant Nokia qui a abandonné son système Symbian OS. Un des points forts de cet OS est la compatibilité avec les produits Microsoft qui équipent 90 % des ordinateurs de la planète tant en terme d'échange de données qu'en terme d'ergonomie. Les développeurs tiers ne sont pas non plus oubliés puisque des solutions logicielles peuvent être conçues à l'aide de Visual C++,

.net ou sous forme de WebApps. Contrairement à Apple, Microsoft laisse une grande liberté aux développeurs puisque les langages et technologies utilisables sont des standards éprouvés et répandus qui sont de plus gratuits d'accès. Les applications ainsi développées sont partageables entre plusieurs personnes sans la nécessité d'utiliser un service intermédiaire tel que l'AppStore d'Apple.

Le langage de programmation

Pour le développement de ses applications, Microsoft privilégie l'écriture en C#. Mais Mais la firme de Redmond ne s'est pas restreinte à une seule façon d'enrichir son catalogue d'application. Les développeurs sont libres de programmer en langage C++ également. Il existe également des applications réalisées avec une version allégée de .NET.

Les contraintes de développement

La principales contraintes des appareils Windows Mobile est la même que pour Android. La diversité des constructeurs et des périphériques fait qu'il est impossible de développer une application compatible à 100% avec tous les appareils.

4.1.1.4. Android, par Google et l'Open Handset Alliance.

En 2005, la firme américaine Google a racheté une start-up du nom de *Android, inc*. [Guignard D., 2011] fondée deux ans plus tôt. Google s'est ensuite allié avec Open Handset Alliance, un consortium de 47 entreprises spécialisées dans l'électronique, le logiciel ou les télécoms, pour distribuer son système d'exploitation basé sur Linux. Ensemble ils voulaient proposer un challenger à l'iPhone qui a bouleversé le monde de la téléphonie. Le premier téléphone équipé du système Android sort en 2008, il s'agit du HTC Dream. Deux ans plus tard, Android était présent sur 26 Smartphones, traduit dans 19 langues et vendu dans 48 pays par 59 opérateurs. Tout comme le BlackBerry, les applications qui s'exécutent sous Android sont développées en Java. La syntaxe est assez proche de celle du langage C++. Créé en 1995, Java est en effet basé sur le C++. Les

équipes de Sun ont simplement choisi de contourner certaines caractéristiques critiques du langage C++, ces caractéristiques qui sont à l'origine des principales erreurs. Cela comprend :

- les pointeurs
- la surcharge d'opérateurs
- l'héritage multiple
- la libération de mémoire est transparente pour l'utilisateur (il n'est plus nécessaire de créer de destructeurs)
- une meilleure gestion des erreurs
- les chaines et les tableaux sont désormais des objets faisant partie intégrante du langage

La contrepartie est une perte en vitesse d'exécution. Java est beaucoup moins rapide que le langage C++ mais ce qu'il perd en rapidité, il le gagne en portabilité. Un code écrit en Java pour s'exécuter sur une JVM installée aussi dans un environnement Windows que dans un environnement Linux ou Mac. Tout comme BlackBerry, c'est une version de Java allégée, déduit aux applications mobiles, qu'il faut utiliser. Il est toutefois nécessaire d'installer le SDK Android pour pouvoir utiliser les API spécifiques telles que :

- Android.app : Cette API implémente tous le modèle d'une application Android. Cela comprend la gestion des start/stop, les activités, les boîtes de dialogue, les alertes et les notifications.

- Android.graphics : Cette API permet la gestion des fichiers de Bitmap, Gif ou JPeg, ainsi que des canvas. Elle offre la possibilité de retravailler ces images à travers des classes telles que Color, Paint, Shader ou TypeFace

- Android.hardware : Grâce aux méthodes de cette API, il devient possible de contrôler et d'utiliser la caméra physique du téléphone.

- Android.utils : On y retrouve toutes les méthodes qui se trouvaient déjà dans Java.utils, à savoir les Objets Log, DebugUtil, TimeUtil et Xml.

- Android.widget : Cet ensemble est nécessaire à tous développement comprenant une interface graphique. Les objets EditText, ListView, FrameLayout, GridView, ImageButton, ProgressBar, RadioButton, RadioGroup, RatingButton, Scroller, TextView et ZoomButton peuvent être intégrer à une Interface Utilisateur pour la rendre plus conviviale.

La liste n'est pas exhaustive, Java Android contient une quarantaine d'API et plus de 700 classes. Ces nombres croissent avec chaque nouvelle version.

De nombreux constructeurs (HTC, Samsung, Motorola, etc) se sont associés à Google pour faire tourner plusieurs de leurs modèles sous Android. Il est donc impossible de développer une application totalement compatible et fonctionnelle avec n'importe quel terminal. Ce grand nombre de modèles différents va apporter une contrainte supplémentaire aux développeurs. Plus une application sera riche, plus il faudra cibler un certain panel de téléphone.

4.1.1.5. Des environnements de développement transverses

Il existe des plateformes de développements capables d'utiliser un projet se basant sur les technologies web (PHP, HTML, JavaScript) pour créer des applications natives android, iPhone, et BlackBerry. On peut citer comme plateformes : Appcelerator Titanium, PhoneGap, et Sencha Touch. La figure 16 présente un comparatif des ces différentes technologies.

Appcelerator Titanium est une plateforme libre pour développer des applications mobiles et de bureau en utilisant des technologies web. Appcelerator Titanium est développé par Appcelerator Inc. et a été présenté en décembre 2008. Elle supporte le développement d'applications pour iPhone (sous Mac OS avec le SDK seulement) et pour Android depuis juin 2009.

PhoneGap est un framework open-source de développement mobile développé par Nitobi Software (racheté par Adobe Systems). Il permet de créer des applications pour appareils mobiles utilisant JavaScript, HTML5 et CSS3, au lieu de langages moins connus telles que l'Objective-C. Les applications qui en résultent sont hybrides, ce qui signifie qu'ils ne sont ni vraiment natif ni purement basée sur le Web.

Sencha Touch est un framework JavaScript spécialement conçu pour développer des applications web mobiles. Ce produit est développé par l'entreprise Sencha, anciennement nommée ExtJS, et qui s'est étendue avec jQTouch et Raphael.

La particularité de ce framework est son développement presque exclusif en JavaScript. Sencha touch est compatible avec les platformes android, iOS (iphone, ipod touch, ipad) ainsi que le tout dernier BlackBerry 6. Il est en particulier adapté à toutes les résolutions d'écran. Il ne s'exécute cependant que sur les navigateurs webkit. Le framework est

disponible en version 1.1 (sortie le 24 mars 2012), sous une licence open source GPL3 et une licence commerciale gratuite

Outil	PhoneGap	Appcelerator Titanium	Sencha Touch
Site de l'éditeur	phonegap.com	appcelerator.com	sencha.com
Licence libre	oui	oui	Oui
Applications cibles	Logiciels embarqués	Logiciels embarqués	Applications Web
Langage de développement	HTML,Javascript	HTML, Javascript, (PHP, Ruby &Python)	HTML5,CSS3,Javascript
Plateformes			
iOS	oui	oui	Oui
Android	oui	oui	Oui
BlackBerry	oui	Beta version	Oui
Windows Mobile	oui	non	Non

Figure 16 - Tableau comparatif des différentes solutions

4.1.2. Les architectures de communication client-serveur

Parmi les technologies de communication de type client-serveur, on peut citer REST, XML-RPC ou encore SOAP, mais encore Java et ses EJB. Il ne faut pas oublier que la solution va intégrer des clients mobiles. Il faut donc optimiser au maximum la quantité de données échangées et leur traitement.

4.1.2.1. La technologie SOAP

La technologie la plus utilisée pour les webservices est SOAP [Chauvet JM., 2002]. Cette dernière s'appuie sur des objets au format XML (Extensible Markup Language) et respecte un protocole de communication RPC (Remote Procedure Call). Le message SOAP est composé de deux parties:

- L'enveloppe contenant le message et les informations concernant l'émetteur et le destinataire.
- Une architecture de données qui définit le format du message et les informations à transmettre.

SOAP est utilisé via HTTP, ceci permet de passer outre les contraintes liées à d'éventuels proxys et pare-feu. L'architecture de données est assez souple pour s'adapter à différents protocoles de transport. Son utilisation est totalement affranchie de la

technologie ou du langage de programmation utilisé. La contrepartie est que la structure à mettre en place pour formaliser l'information est plus lourde que d'autres technologies comme REST par exemple. La structure étant fixée lors de l'élaboration du protocole, une modification de cette dernière impose des adaptations du côté client comme du côté serveur du fait qu'un fort couplage est maintenu entre le client et le serveur.

4.1.2.2. La technologie REST

REST (REpresentational State Transfer) [Richardson L. et Ruby S., 2007] part du principe que le protocole HTTP suffit largement à l'ensemble des besoins d'un service Web. Il utilise donc les méthodes GET, POST, mais aussi PUT, DELETE, CONNECT. On se connecte uniquement à une adresse HTTP pour récupérer les informations. Les formats utilisés pour REST sont le XML (Extensible Markup Language) et JSON (JavaScript Object Notation). JSON est un format de données textuel qui représente de l'information structurée. Ses principales caractéristiques sont :

- il est simpliste, ce qui le rend lisible aussi bien par un humain que par une machine
- sa syntaxe est réduite et non extensible, il est donc facile à apprendre
- ses types de données sont connus et simples à décrire (tableau, booléen, chaines,…)
- il n'y a pas besoin de le *parser* (parcourir) pour récupérer une information précise

De son côté, les caractéristiques du XML sont :

- la lisibilité : aucune connaissance n'est nécessaire pour comprendre le contenu d'un document XML
- il est autodescriptif et extensible. Cela signifie que sa structure est décrite dans le message lui-même, elle peut être adaptée et évoluer sans modifier quoique ce soit chez le client ou le serveur.
- universalité et portabilité : les différents jeux de caractères sont pris en compte
- extensibilité : un document XML doit pouvoir être utilisable dans tous les domaines d'applications

4.1.2.3. Les objets Enterprise Java Bean (EJB)

Les EJB [Lafosse J., 2011] sont une suite de composants développés en Java côté serveur. C'est une spécificité de J2EE. Cette architecture propose un cadre contenant des objets distribués (déployés sur des serveurs distants) hébergés sur un serveur d'application. Il existe différents types d'objets qui peuvent représenter des données (on parle alors d'EJB Entité), des services (on parle alors d'EJB Session), ou encore des tâches dans le cadre d'une communication asynchrone. Dans ce dernier cas on parle d'EJB Message. Le serveur d'application se charge de la gestion des objets (création, destruction, activation). Le client exécute des appels RMI pour appeler des EJB et utiliser leurs méthodes ou leurs attributs.

4.1.2.4. L'architecture Remote method invocation (RMI)

RMI [Grosso W., 2001] [Fron A., 2007] est une interface de programmation Java utilisée pour l'appel d'objets ou de méthodes distantes. Grâce à cette technologie, il devient possible de réaliser des communications via HTTP entre des objets Java distants, c'est-à-dire s'exécutant sur des machines virtuelles Java distinctes. Le principal avantage de RMI est qu'il occulte complètement la communication client-serveur aux yeux du développeur. La conception d'applications distribuées en est donc simplifiée. Son utilisation impose toutefois d'utiliser des machines virtuelles Java sur le client et sur le serveur.

4.1.2.5. Le composant Asynchronious Javascript and XML (Ajax)

Ajax [Catteau B. et Faugout N., 2009] est une architecture permettant de construire des applications web et des sites web dynamiques et interactifs sur le poste client. Elle combine plusieurs technologies telles que les CSS, le Javascript, les XML ou le JSON. La plus-value d'Ajax est une diminution de latence, l'apport de nouvelles fonctionnalités et l'augmentation de la réactivité d'une application web. Alors que des technologies concurrentes nécessitent l'installation d'un Plug-in sur le poste client, Ajax utilise les technologies présentes de série sur la plupart des navigateurs internet (Internet Explorer, Mozilla Firefox, Google Chrome). Dans une application Web, la méthode classique de dialogue entre un navigateur et un serveur est la suivante : lors de chaque manipulation faite par l'utilisateur, le navigateur envoie une requête contenant une référence à une page Web, puis le serveur Web effectue des calculs, et envoie le résultat sous forme d'une nouvelle page Web à destination du navigateur. Celui-ci affichera alors la page qu'il vient de recevoir. Chaque manipulation

entraîne la transmission et l'affichage d'une nouvelle page et l'utilisateur doit attendre l'arrivée de la réponse pour effectuer d'autres manipulations. En utilisant Ajax, le dialogue entre le navigateur et le serveur se déroule la plupart du temps de la manière suivante : un programme écrit en langage de programmation JavaScript, incorporé dans une page web, est exécuté par le navigateur. Celui-ci envoie en arrière-plan des demandes au serveur Web, puis modifie le contenu de la page actuellement affichée par le navigateur Web en fonction du résultat reçu du serveur, évitant ainsi la transmission et l'affichage d'une nouvelle page complète.

4.2. Choix technologiques

4.2.1. Choix du Smartphone cible

Les choix de la plateforme cible se base essentiellement sur une analyse de marché réalisée par Thomas Estimbre pour le site internet [Estimbre T., 2011]. Cette étude compare les parts de marché des différents systèmes d'exploitation mobile:

4.2.1.1. Windows Mobile, une santé fragile

La part de marché de Microsoft s'est nettement dégradée en 2011 pour arriver à 2% (figure 17). La société compte sur son nouveau partenariat avec Nokia pour redresser la barre. Nokia a en effet annoncé l'abandon de son système d'exploitation Symbian pour le remplacer par celui de Microsoft, Windows Mobile 7. Cette annonce est récente et le lancement des téléphones Nokia équipés du système d'exploitation de Microsoft est nuancé par le succès des systèmes concurrent. A ce jour, Windows Mobile peine encore à pénétrer le marché.

4.2.1.2. BlackBerry, la chute

BlackBerry est en chute libre. Les changements opérés à la tête du groupe RIM (remplacement de Mike Lazaridis et Jim Balsillie, les deux co-dirigeants historique, par Thorsten Heins), l'échec de leur tablette Playbook et les soucis techniques au niveau mondial sur leurs serveurs de communication à l'automne 2011 n'annoncent rien de bon.

La technologie évolue, les téléphones proposés par les marques concurrentes sont de plus en plus puissant et de plus en plus fiable et arrivent maintenant à séduire un public plus large composé de particuliers et de professionnels. Les opérateurs mobiles contribuent également (malgré eux) à la chute de BlackBerry. Les forfaits proposés aujourd'hui vendent des débits internet vraiment intéressant et des offres data quasi illimitées. Ce qui faisait la force de BlackBerry, à savoir des téléphones fiables, des claviers permettant la saisie aisée de messages, les agendas et les mails récupérés sur le téléphone en temps réel, n'est plus uniquement la marque de la société Canadienne.

4.2.1.3. Android et iOS, les challengers

Apple a réussi à se faire une bonne place dans le marché des technologies mobiles et est directement concurrencé par Android. Les deux acteurs s'affrontent sur tous les continents aussi bien sur le marché des Smartphones que sur celui des tablettes. Apple a réussi à imposer son image à travers des produits innovants et techniquement très évolués. Le système d'exploitation iOS propose une interface intuitive, fluide et qui réagit très rapidement. La firme de Cupertino a réussi à conquérir 18% de parts de marché en 2011 (figure 17). Google s'est associé à plusieurs constructeurs de téléphone pour installer son système d'exploitation sur un vaste panel d'appareil. Si Android n'a pas la même image que les produits Apple, sa force réside dans le fait qu'il est installé sur des appareils couvrant une large gamme de prix. Ceci lui permet donc de toucher un très large public composé aussi bien de technophiles que de néophytes et de regrouper à lui seul 43% des parts de marché en 2011 (figure 17). Ce sont donc ces deux plateformes qui seront étudiées pour mon projet. Il est préférable de cibler les OS prédominants avec une approche orientée qualité, plutôt que tous les OS de manière passable C'est pour cela que j'essayerai de développer une solution pour Android puis une solution pour iPhone. Ceci sera privilégié par rapport à une solution transverse.

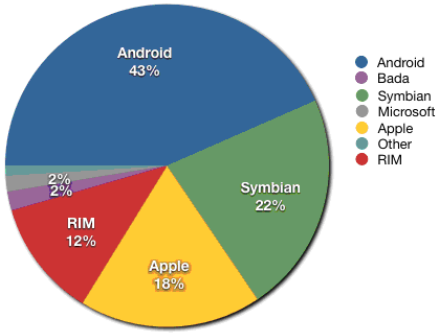

Figure 17 - Les parts de marché des différents OS en 2011

4.2.2. Choix du Webservice

En ce qui concerne le développement du serveur d'applications, il faut choisir une technologie qui nous permet d'obtenir un produit final robuste et fiable. Ce résultat peut être obtenu avec l'utilisation de Webservice [Maesano L., 2003]. Les principaux avantages de cette technologie sont :

- l'interopérabilité entre différentes plateformes,
- l'utilisation de standards et protocoles ouverts,
- l'utilisation du protocole http, permettant de contourner les pare-feu,
- les échanges peuvent être sécurisés simplement en utilisant SSL, ou de manière plus approfondie avec WS-Security (et autres spécifications de sécurité WS-*). On peut alors avoir une garanti sur l'identité de l'utilisateur, ainsi qu'une non-répudiation grâce à un système de signature

Le principal inconvénient est la faible performance du protocole. Une solution alternative a été envisagée. Il s'agit de la possibilité d'utiliser les EJB (Enterprise JavaBeans) de Java qui fournissent un cadre pour des composants distribués. Des objets développés sur le serveur d'applications sont tout à fait exploitable à distance sur un site client ou sur une application Smartphone. Cette technologie offre de bonnes performances. Toutefois, il faut noter quelques inconvénients :

- la nécessité d'ouvrir des ports pour communiquer (problèmes de pare-feu ou de proxy)
- l'utilisation obligatoire de java pour le client (serveur web et application Smartphone)
- la gestion manuelle de la sécurité des échanges

Le choix a surtout fait à partir du critère de l'interopérabilité entre les systèmes. En effet, si le serveur d'applications est conçu à la manière d'un service web, le site client ainsi que l'application Smartphone pourront être développés à partir de n'importe quelle technologie (java, php, python, ruby, etc.). Dans le cas contraire, tout devra être réalisé en Java pour pouvoir faire appel aux EJB via RMI. L'utilisation des EJB sera donc écartée pour gagner en simplicité lors de l'intégration de notre solution sur les sites clients.

4.3. Gestion du projet

Le sujet de ce stage Ingénieur a été proposé à Stéphane Vialle, Responsable régional de la spécialité informatique au CNAM de Lorraine, au mois d'Octobre 2011. Les différentes étapes qui m'ont amené à ce mémoire sont reprise dans la figure 18 :

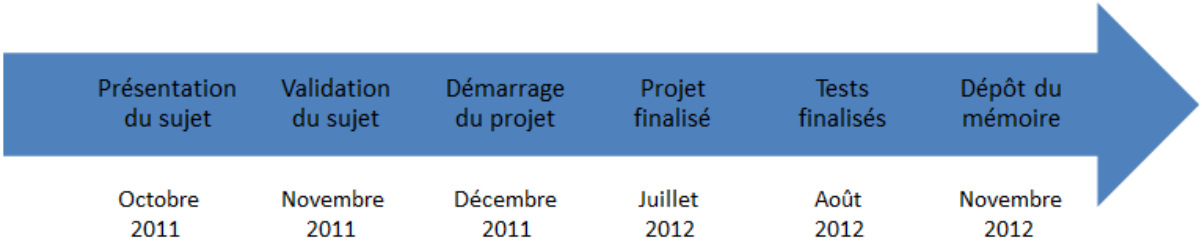

Figure 18 - Chronologie du projet

Le projet s'est déroulé dans les locaux d'InTech S.A. au 17-19 avenue de la Libération à Schifflange au Luxembourg. Durant toute cette période, j'ai été encadré par Antoine DETANTE, Consultant au sein de l'entreprise. La partie Conception et Développement s'est déroulée sur une période de huit mois (jusqu'en juillet 2012). Ensuite je me suis accordé un mois pour éprouver le système, analyser ses limites et tester mes applications. Durant cette période j'ai eu la chance de pouvoir apporter une dimension encadrement à mon travail. Ma société m'a en effet confié la responsabilité du stage de fin d'étude de Ludovic IGGIOTTI, étudiant en 5ème année à l'Ecole Supérieure des Sciences et Technologies de l'Ingénieur de Nancy (ESSTIN). Les différentes étapes réalisées lors de ces neuf mois sont reprises dans les figures 19 et 20.

Tâches
A. Analyse du marché
B. Proposition d'une solution.
C. Recherche d'acteurs implantés sur le marché.
D. Etat de l'art de l'authentification et des technologies mobiles
E. Comparatif (avantages/inconvénients) des technologies candidates
F. Réalisation d'un prototype
G. Déploiement et évaluation

Figure 19 - Etapes du projet

Nom de la tâche	Date de début	Date de fin	Durée
A. Analyse du marché	01/11/11	30/11/11	20
B. Proposition d'une solution	30/11/11	13/12/11	10
C. Recherche d'acteurs implantés sur le marché	30/11/11	13/12/11	10
D. Etat de l'art de l'authentification et des technologies mobiles	30/11/11	13/01/12	30
E. Comparatif (avantages/inconvénients) des technologies candidates	13/01/12	26/01/12	10
F. Réalisation d'un prototype	26/01/12	31/07/12	134
G. Déploiement et évaluation	01/08/12	31/08/12	23

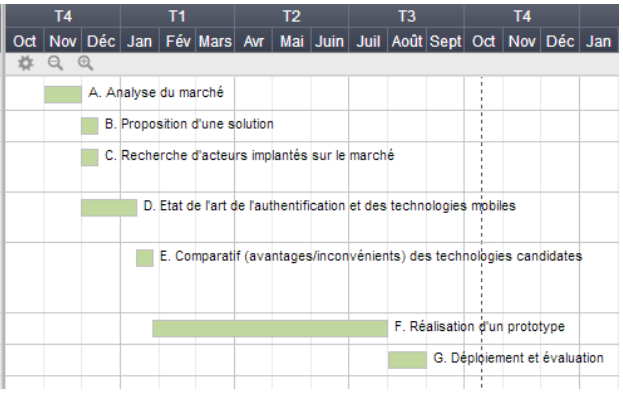

Figure 20 - Ordonnancement des phases du projet

Les premières semaines du projet ont été consacrées à une analyse du marché pour accentuer la pertinence du sujet qui avait été validé. Ce travail a révélé l'émergence d'un besoin croissant de sécurité dans le domaine de la mobilité et du nomadisme. Le résultat a fait évoluer le sujet vers la conception d'une solution de connexion alliant une authentification forte depuis un terminal mobile. En m'inspirant du principe de l'authentification SSO (Single Sign-On), J'ai voulu élaborer une solution permettant de se connecter à plusieurs sites internet en centralisant l'authentification sur un seul serveur. La plus-value que je souhaitais apporter était de remplacer la traditionnelle saisie d'identifiants par un moyen plus convivial. L'utilisation des QR codes (Quick Response Code) s'est imposée vu la pénétration de cette technologie dans notre vie quotidienne.

Couplée au sujet initial qui consistait à porter la couche sécurisée de LuxTrust sur les Smartphone, la solution que j'ai proposée offre une possibilité de connexion simple et conviviale en intégrant tous les avantages d'une authentification forte LuxTrust. Cette solution a ensuite été présentée à LuxTrust qui était intéressé par un portage de leur Common Layer sur des plateformes mobiles de type Smartphone ou tablette. J'ai également rencontré des responsables des développements pour la société Mobey. Etant implanté sur le Luxembourg avec leur produit FlashIz, solution de paiement mobile par QR Code, la société souhaite faire évoluer son site internet et en particulier la façon de s'y connecter.

Avant de me lancer dans les développements, j'ai réalisé un état de l'art des différentes technologies impliquées dans le projet. J'ai comparé les différents systèmes d'exploitation pour périphérique mobile avec leurs points forts et leurs lacunes puis je me suis renseigné sur les contraintes à respecter pour développer des applications sur

chacun de ces systèmes. J'ai également passé du temps à me documenter sur le monde du chiffrement de données avec ses algorithmes de cryptage et ses fonctions de hashage. Avec ce bagage, j'étais armé pour analyser et comprendre le fonctionnement du Common Layer. En fin février, Ludovic m'a rejoint sur le projet et nous avons travaillé ensemble pour concrétiser l'architecture de ma solution. Ma façon de travailler a nettement évoluée du fait qu'il a fallu faire avancer la partie portage du Common Layer sur Android tout en encadrant rigoureusement mon stagiaire. En effet, il ne suffisait pas de lui donner des consignes mais il était bien nécessaire de discuter avec lui de sa vision de l'objectif et de ses idées pour attendre ce dernier. Durant les quatre mois qui ont suivi, je l'ai guidé dans le but de faire évoluer nos prototypes afin qu'ils respectent le cahier des charges que je m'étais imposé. Nous nous sommes astreints à programmer régulièrement des mises au point durant lesquelles je m'assurais qu'il suivait le bon chemin en utilisant les outils dont nous avions discuté ensemble.

Durant le mois de juillet, nous sommes entrés dans la phase de finalisation du projet. Ludovic s'est plongé dans l'analyse du site existant de FlashIz afin de trouver un moyen d'intégrer notre librairie de connexion au travers d'un QR Code. La solution étant pleinement fonctionnel dans un environnement réel, je me suis chargé du développement d'une application simple sur iOS. Cette application devait pouvoir scanner un QR Code pour valider une connexion à un site internet mais sans implémenter la couche d'abstraction qu'est le Common Layer. Le point à valider était l'utilisation de librairies écrites en C dans une application développée en Objective-C, mais ceci sera expliqué plus loin dans ce mémoire. Le mois d'août a été consacré aux tests du système. J'ai réalisé des tests applicatifs mais également des tests unitaires et des tests de couverture. Une analyse des limites de la solution et de ses vulnérabilités a également été faite. Ceci est détaillé dans un paragraphe qui va suivre. Je me suis limité à une étude théorique pour me laisser le temps de rédiger le mémoire afin de le rendre dans les délais qui m'avaient été accordés. En effet, si la rédaction du mémoire s'est faite en parallèle de mes autres tâches, elle a surtout été finalisée durant les mois de septembre et octobre 2012.

Les délais tenus lors de ce projet sont un peu différents de ceux estimés dans le planning prévisionnel. Afin de valider la pertinence de la solution sur le plan technique et le plan marché, les six premières semaines ont été consacrées à une analyse de marché et à l'identification d'acteurs bien implantés sur la place Luxembourgeoise. L'objectif original a été étoffé avec la proposition d'une solution allant au-delà de la simple implémentation du

Common Layer sur un Smartphone. Le portage du Common Layer sur un environnement Android a été réalisé avec un mois de plus que les délais prévus. Le prototype d'application capable de communiquer avec les serveurs de LuxTrust a été finalisé à la fin du mois d'Avril avec deux mois de retard sur l'échéance prévue. Les deux mois qui ont suivi ont été consacrés à la finalisation de toute l'architecture. A la fin du mois de juin, l'application Android pouvait scanner un QR Code affiché par le dernier prototype du site client et contacter LuxTrust pour valider son authentification. Mais le portage du Common Layer sur iPhone, qui devait initialement débuter en début avril, avait pris trois mois de retard sur le planning prévu. Nous avons alors réalisé une application simple permettant de scanner le QR Code et de faire appel à notre webservice. Le portage du code Java vers Objective-C (le langage de programmation utilisé pour iOS) a, lui, été remis à une remis à plus tard. En ajoutant un mois pour réaliser des tests applicatifs et des tests unitaires, la finalisation du mémoire a pu se faire avec un mois de décalage par rapport à la date prévue.

4.4. Conception de l'application

4.4.1. Description de l'architecture matérielle

La figure 21 représente les différents éléments qui composent l'architecture matérielle.

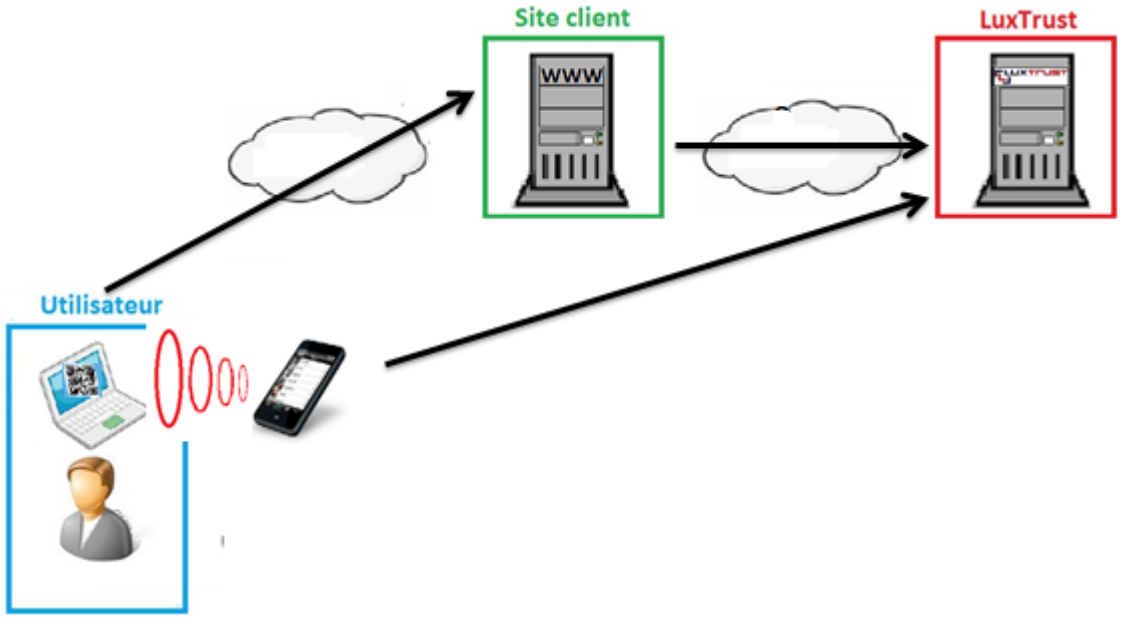

Figure 21 - Schéma de la solution

L'architecture se compose de plusieurs acteurs qui vont chacun jouer leur rôle dans l'authentification. L'utilisateur va afficher un site internet sur son ordinateur. Pour cela n'importe quelle machine ayant accès à internet pourra convenir. Il n'existe aucun prérequis pour ce niveau de l'architecture. Il réalisera une demande de connexion pour afficher une partie privée du site internet. Le serveur Web devra intégrer notre librairie pour pouvoir communiquer avec les serveurs de LuxTrust en respectant le protocole mis en place. Il sera ainsi capable de recevoir le jeton pour le transformer en QR Code à afficher sur l'écran de l'utilisateur. Cette librairie a été écrite en Java J2EE. Le site client doit donc être capable d'intégrer et d'utiliser des fichiers jar. Il est toutefois envisageable de faire évoluer la partie "site client" pour la porter vers d'autres technologies. Cette souplesse est permise par l'utilisation de standards et protocoles ouverts dans une architecture orientée services.

Concernant le serveur d'authentification intégré aux serveurs de LuxTrust, il s'agit d'une librairie Java J2EE hébergée par un serveur d'application. Dans un premier temps, la version que nous avons déployée va se charger de générer puis de transmettre les jetons aux serveurs web puis d'envoyer un signal à ces derniers lorsque l'utilisateur s'est correctement authentifié. LuxTrust n'a pas souhaité nous communiquer leur architecture de base de données. A ce jour, la partie qui gère le jeton et celle qui gère l'authentification par Common Layer sont distincte. La fusion de ces deux parties sera prise en charge ultérieurement par d'autres équipes de développement habilitées. Le dernier acteur de cette architecture est l'application Smartphone. Nous proposerons une version distribuée sur Google Play, le marché d'applications Android, et sur une autre sur l'Apple Store, celui de Apple. La version Android a été testée sur des versions 2.1 et 2.3 du système d'exploitation. Dans le cadre d'un déploiement à grande échelle, il faudra se poser la question de la compatibilité avec les versions plus récentes du système. Le cas est plus simple pour la version iPhone. En effet, les différentes versions du téléphone utilisées aujourd'hui (3, 3GS, 4 et 4S) tournent sous la version 5 de l'iOS. La question de la compatibilité se posera certainement lorsque la version 6 se sera démocratisée.

4.4.2. Modélisation du fonctionnement de l'application

Le fonctionnement de l'application est représenté par la figure 22.

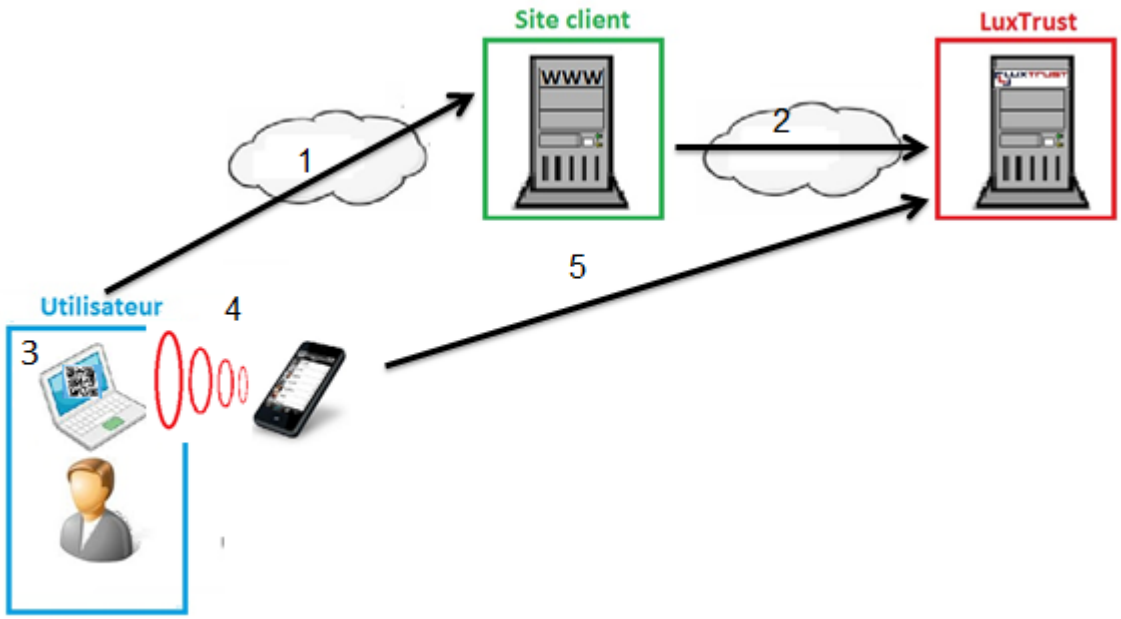

Figure 22 - Listes des différentes étapes du processus

Les différentes étapes sont :

1. L'utilisateur va afficher le site internet sur son ordinateur avec le navigateur de son choix. Il va simplement demander la connexion en cliquant sur un lien ou sur un bouton.

2. Le serveur web hébergeant le site internet va effectuer un appel vers les serveurs de LuxTrust pour demander un jeton (chaine de caractère) pour identifier la session.

3. Cet appel va lancer un processus durant lequel LuxTrust va générer un identifiant unique et le transmettre au serveur web qui va ensuite l'afficher dans le navigateur internet de l'utilisateur sous forme de QR Code.

4. A ce stade, l'utilisateur se retrouve face à un QR Code à scanner, ce dernier correspondant à un numéro unique connu chez LuxTrust. Il lui reste donc à le scanner à partir de l'application dédiée installée sur son Smartphone.

5. Les données (jeton, identifiants de connexion et mot de passe) sont donc transmises à LuxTrust de façon sécurisée à travers le Common Layer.

Pendant ce temps, le site internet reste toujours en attente pour laisser le temps à l'utilisateur un délai nécessaire pour s'authentifier. Lorsque LuxTrust a validé l'authentification, un message est envoyé à l'application mobile et au site internet. L'application traite ce feedback pour indiquer à l'utilisateur que l'authentification est validée, la page du site internet se met à jour pour donner l'accès à son contenu. Les communications entre les serveurs de LuxTrust et le site Internet (lors de la transmission du jeton, puis de la confirmation de l'authentification) doivent bien évidemment se faire au travers d'une connexion sécurisée avec authentification mutuelle (SSL) : ces deux échanges sont très importants dans le protocole d'authentification, et il est nécessaire de s'assurer de l'identité des deux acteurs (LuxTrust et le site Internet d'origine) pour éviter des interceptions ou des injections malveillantes. Avec cette solution, l'utilisateur devient réellement acteur de son authentification et n'est plus un simple spectateur. Cette opération se réalisant entre trois membres, la sécurité est accrue. Il sera difficile, pour une personne malveillante, de venir récupérer les informations nécessaires pour pirater le compte. Le diagramme de la figure 23 modélise un cas où l'identification se fait avec succès.

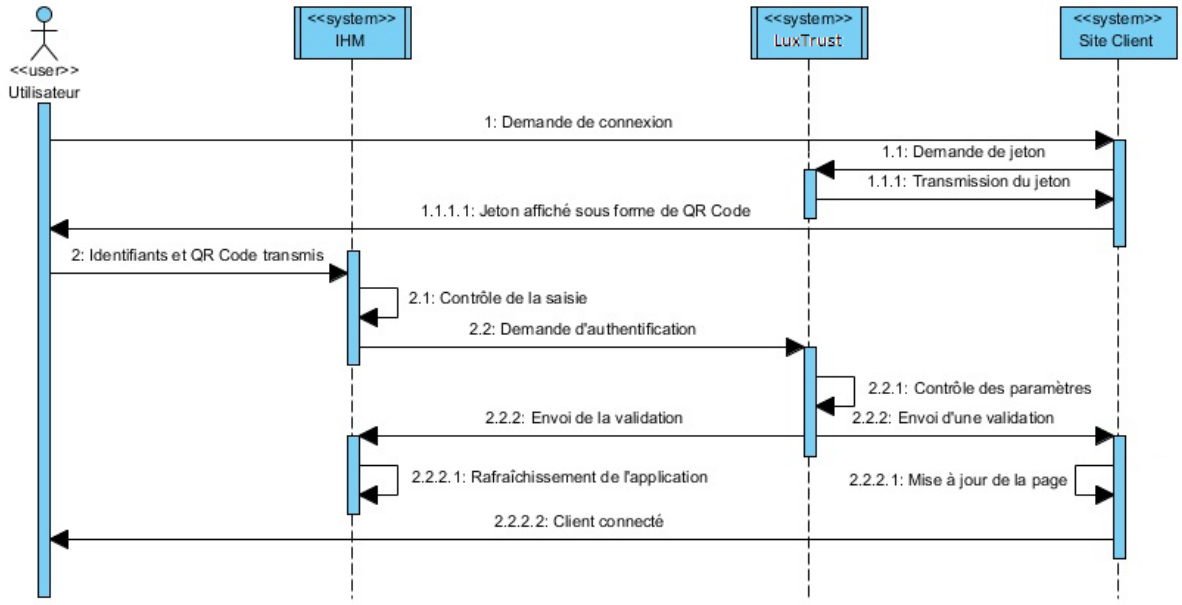

Figure 23 – Modélisation du fonctionnement de l'application

L'utilisateur va effectuer une demande de connexion sur le site client. Ce dernier contacte alors LuxTrust pour demander un jeton qui sera affiché sur le site sous forme de QR Code. Lorsque l'IHM reçoit les identifiants de l'utilisateur et que le jeton est scanné, elle vérifie juste que les informations ne sont pas vides puis transmets le tout à LuxTrust via le Common Layer réimplémenté pour être compatible avec les technologies mobiles. LuxTrust va valider l'authentification et transmettre aux deux systèmes une confirmation. L'interface IHM va donc se mettre à jour pour informer l'utilisateur du bon déroulement de l'opération, et le site client va rafraichir sa page pour autoriser l'accès au contenu protégé. Le diagramme de la figure 24 modélise les différents états que peuvent prendre les deux systèmes liés à l'authentification, le serveur web gérant le site client et le serveur d'authentification installé chez LuxTrust.

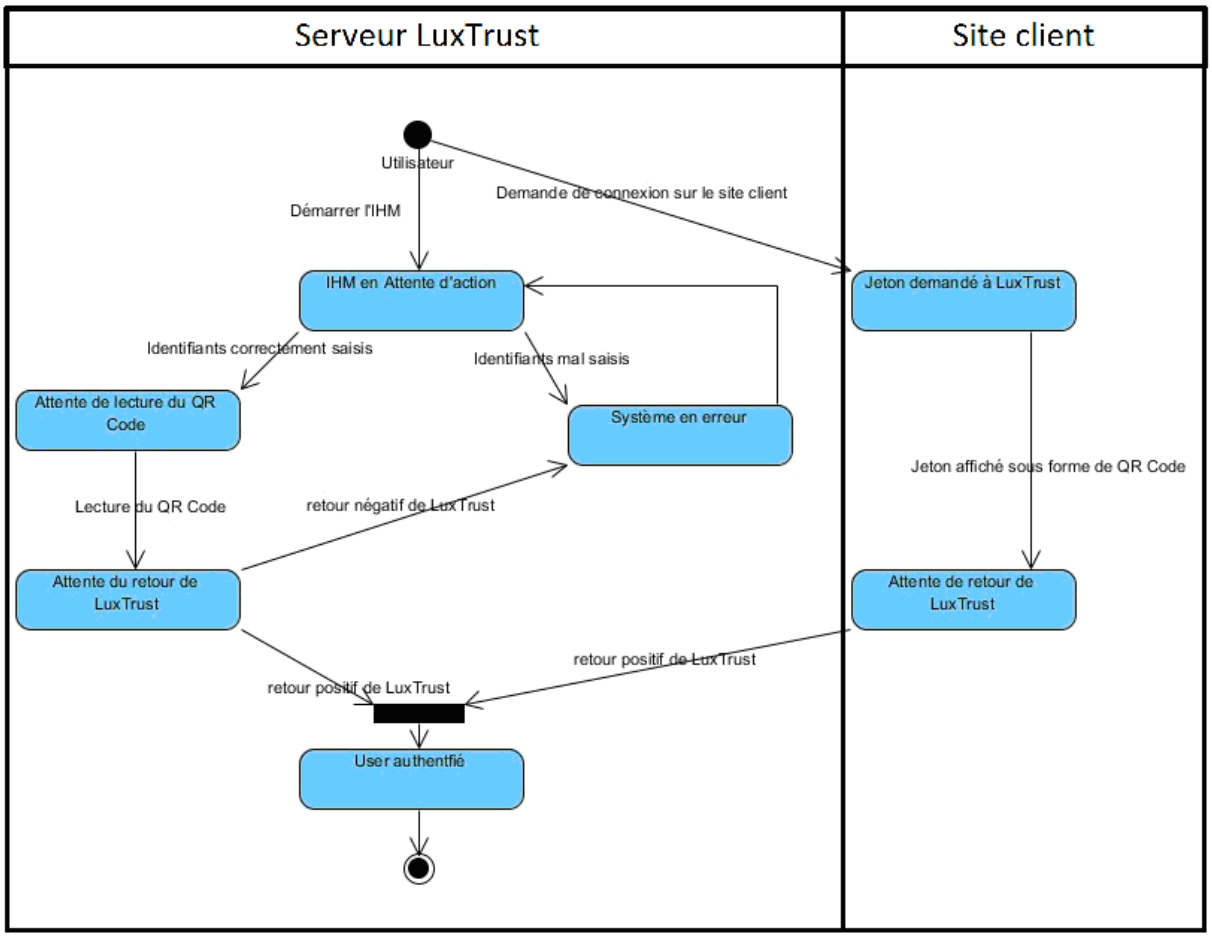

Figure 24 – Modélisation du processus d'authentification

Au démarrage, l'interface IHM est en attente d'une action de l'utilisateur. Si les identifiants sont correctement saisis, l'application va scanner un QR Code. Dans le cas contraire, le système est en erreur, informe l'utilisateur et revient à son statut initial. Le jeton lu au travers du QR Code est transmis à LuxTrust avec les paramètres de connexion puis le système se met en attente d'un retour de ce dernier. Afin d'arriver à l'état final dans lequel l'utilisateur est authentifié, il faut que LuxTrust ait répondu favorablement à l'IHM et au serveur Web du site client.

La figure 25 est un workflow qui modélise le fonctionnement de l'ensemble du système et les liens entre les différents états des acteurs.

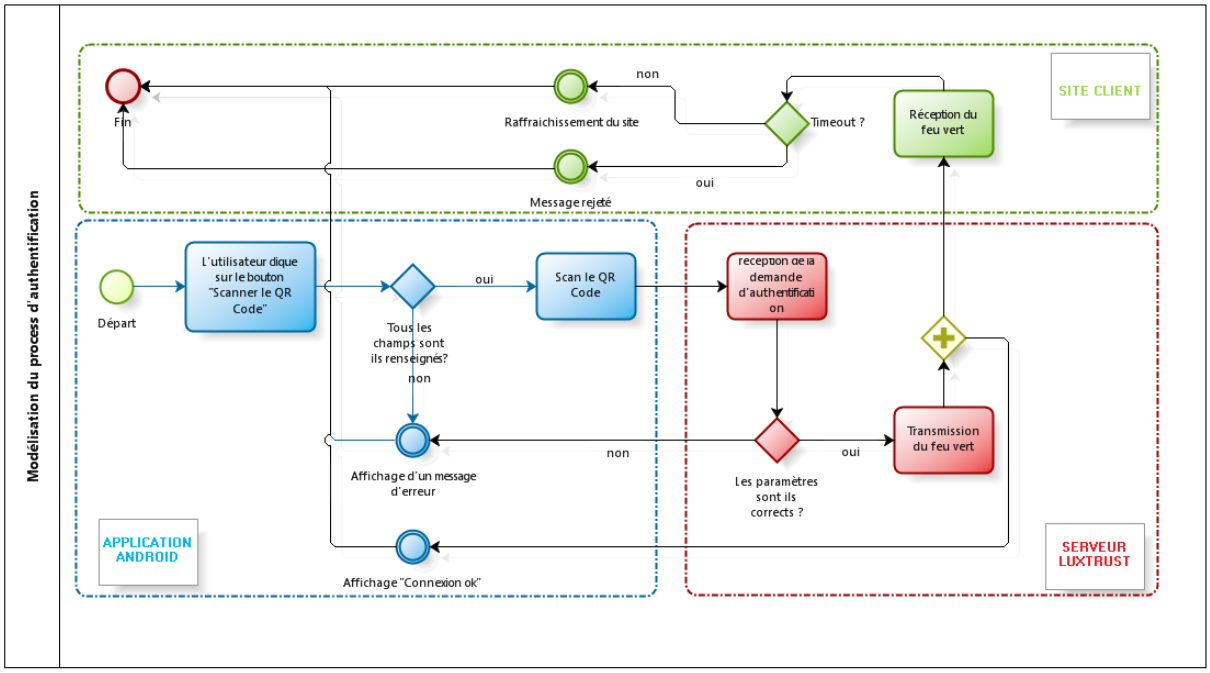

Figure 25 – Description des différentes actions réalisées par les éléments de l'architecture

Les différents éléments de l'architecture sont représentés en différentes couleurs sur le workflow illustré par la figure 25. En bleu, il s'agit de l'interface IHM des applications Smartphone. La librairie qui sera à intégrer chez LuxTrust est en rouge. Enfin on symbolise en vert le comportement du site client. Le processus représenté ici est le schéma classique d'une demande d'authentification. En fonction des états de chaque acteur, l'ensemble du système va évoluer vers un état final dans lequel l'utilisateur est authentifié ou non.

4.5. Réalisation

4.5.1. Contraintes de réalisation

La solution décrite précédemment ne pourra pas être mise en place sans quelques adaptations au préalable. Le serveur d'authentification qui va générer les jetons et transmettre un signal au serveur web ne pourra pas accéder aux bases de données de LuxTrust pendant la phase de conception et de test. L'idée est donc de dissocier l'appel au Common Layer qui va valider l'authentification et l'envoi du signal au serveur web. Dans la version finalisée de la solution, l'application Smartphone va scanner le QR Code et contacter

LuxTrust au travers du Common Layer. C'est ce module intégré chez LuxTrust qui enverra un signal de validation au Smartphone mais également un signal de validation au serveur web qui gère le site client. La figure 26 représente la solution adaptée pour nos besoins pendant toute la phase de conception.

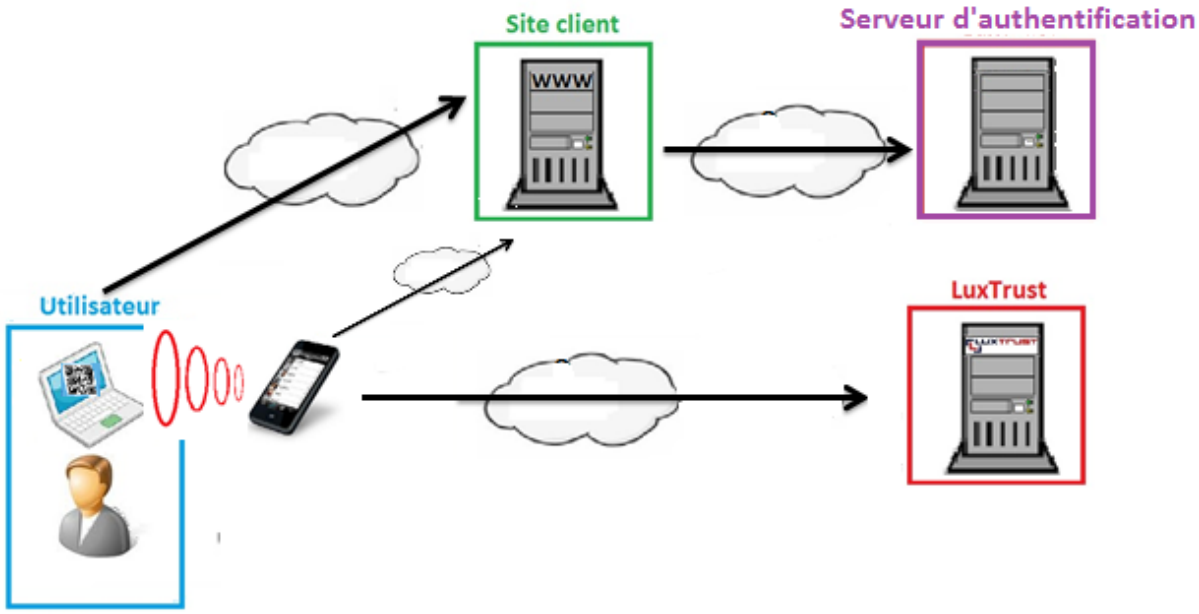

Figure 26 - Description de la solution adaptée

Lorsque l'utilisateur va se connecter au site internet, le serveur web contacte le serveur d'authentification pour demander le jeton à afficher. Ceci ne diffère pas de la solution présentée plus tôt. La différence sera notée lors de l'authentification. L'application Smartphone va scanner le QR Code puis contacter les serveurs LuxTrust au travers du Common Layer. Si l'authentification est réussie c'est l'application Smartphone qui va envoyer le signal de validation au serveur web qui rafraichira alors la page dans le navigateur de l'utilisateur.

4.5.2. Verrous technologiques

Le premier verrou technologique identifié est la compatibilité entre le Common Layer et les technologies mobile. L'application est développée en Java standard. Or les versions mobiles sont développées en J2ME (Java 2 Micro Edition). L'utilisation des fichiers jars dans un projet Android ne pourrait pas fonctionner car J2ME est une version "allégée" de Java et toutes les fonctions ne sont pas forcément disponibles. Il faudra donc analyser le

code et identifier les composants nécessaires au processus d'authentification. Si certains de ces composants ne sont pas compatibles avec J2ME, il faudra chercher un moyen de contournement ou une solution similaire.

Le second verrou va se poser lors du développement d'une application sous iOS. Le système d'exploitation d'Apple tourne sous un noyau Unix et les applications sont écrites dans un langage dérivé du C et du C++ appelé Objective-C. Dans un premier temps il va être nécessaire de porter le code depuis Java vers Objective-C, dans un second temps il faudra voir si les éventuelles librairies utilisées dans la version Java peuvent l'être dans une version Objective-C. Si le Common Layer devient utilisable sur des périphériques mobiles, il sera alors possible de bénéficier des avantages d'une authentification forte directement sur le téléphone portable de l'utilisateur.

Il est possible aussi d'imaginer d'autres usages, à terme : par exemple, signature ou chiffrement des mails directement depuis l'application Mail native de l'OS mobile, car on a une librairie de chiffrage/signature utilisable sur le mobile. Voir des applications pour encrypter le contenu du Smartphone. Lorsque les applications mobiles seront développées, il faudra mettre en place une architecture logicielle pour permettre au serveur Web de communiquer avec les serveurs d'Authentification de LuxTrust mais également de rafraichir la page internet du navigateur de l'utilisateur. Il va falloir trouver une solution légère qui s'intègre facilement dans les produits déjà en place mais une solution suffisamment évoluée techniquement pour ne pas rebuter l'utilisateur. Il s'agit là du dernier verrou technologique identifié. Il sera toutefois important de le lever car il va impacter l'intérêt de l'utilisateur pour notre solution mais également la bonne volonté des partenaires qui l'intègreront à leurs systèmes. En effet, il va falloir faire les bons choix techniques pour proposer une solution conviviale et esthétique qui ne rebute pas l'utilisateur tout en permettant au serveur Web de rafraichir le navigateur du client sans surcharger ni le réseau ni la machine.

4.5.3. Difficultés du projet

L'API du Common Layer n'est pas compatible avec Java J2ME et encore moins compatible avec du code Objective-C. Il faut donc analyser l'API existante pour effectuer un portage sur Android et iOS. Le premier pas était de développer une petite applet Java qui communiquera avec un serveur de test installé chez LuxTrust et qui utilise les JAR fournis par LuxTrust. La première étape a donc consisté à faire de la rétro conception sur les

fichiers jar du Common Layer pour pouvoir le personnaliser et lui indiquer la bonne adresse de destination. Il faudra ensuite reprendre les Objets des librairies Cryptomathic pour s'en affranchir et les réécrire si nécessaire pour qu'ils soient compatibles avec Java Mobile (J2ME). L'intérêt de réimplémenter les différents sources du Common Layer et qu'il devient ensuite possible de porter le code vers un autre langage tel que l'Objective-C. Les outils utilisés par Cryptomathic pour le chiffrement sont déjà une réécriture des classes de Bouncy Castle. Cette librairie ne semble pas, à ce jour, compatible avec le SDK Android. Il va falloir également trouver une solution de contournement.

Une fois ces verrous technologiques levés, il conviendra de développer le module à intégrer aux Serveur LuxTrust ainsi qu'une libraire utilisable par un client qui souhaite l'intégrer à son site client. C'est là que vont se poser plusieurs questions sur le choix de la technologie à utiliser afin de ne pas saturer les réseaux en communications inutiles ou de surcharger le site client avec des composants trop lourds.

4.5.4. Le Common Layer

4.5.4.1. Présentation

Le Common Layer est une API Java utilisée comme interface entre l'application cliente et les serveurs de Luxtrust pour le produit Signing Server. Ce Common Layer implémente la *Crypto Token Interface* développée par Cryptomathic et se compose de plusieurs modules dont l'ensemble est schématisé par la figure 27.

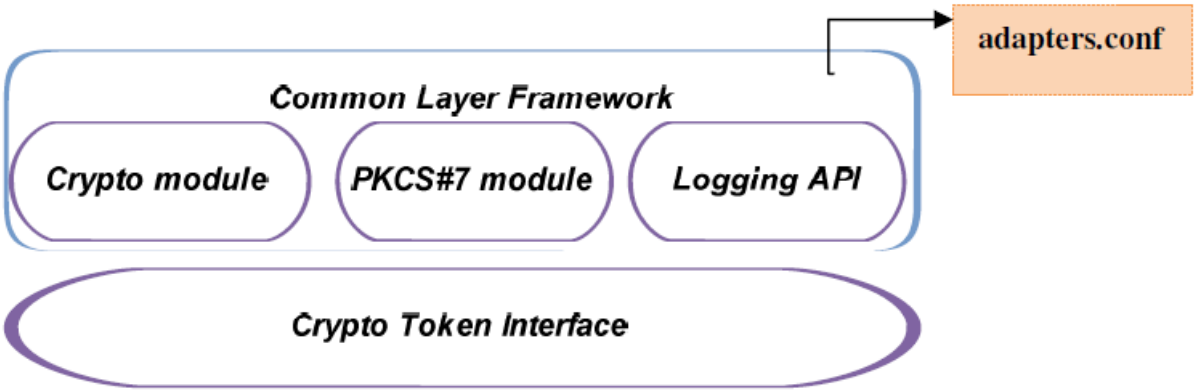

Figure 27 - Description de la couche Common Layer

Les différents composants sont :

- Crypto module : Ce module fournit tous les outils de chiffrement ainsi que les fonctions de hashage.
- PKCS#7 module : Il permet de générer des signatures électroniques qui respectent les standards PKCS #7.
- Logging API : Cet API regroupe toutes les méthodes de Logging (DEBUG/WARN/ERROR) qui sont appelées lors de l'utilisation de l'application.
- adapters.conf : il s'agit d'un fichier properties de configuration faisant par exemple le lien entre le système d'exploitation utilisé et les classes à utiliser pour implémenter les interfaces Java.

Si le cadre d'utilisation du Common Layer diffère (autre système d'exploitation, autre produit LuxTrust) il serait normalement nécessaire de modifier le code du Common Layer pour l'adapter à la nouvelle situation. Or l'utilisation d'une interface couplée avec un fichier de properties (adapters.conf) permet de s'affranchir de ces modifications. La Crypto Token Interface est une interface standard commune à tous les produits de LuxTrust utilisables avec les systèmes les plus populaires. Elle sert de pont entre l'application et les services proposés par les serveurs LuxTrust. Les équipes de développement qui souhaitent intégrer le Common Layer à leurs systèmes n'ont donc pas besoin d'en connaitre le fonctionnement. Il suffit que LuxTrust leur fournisse la documentation d'utilisation de l'interface avec les différentes méthodes à appeler pour qu'ils soient capables d'adapter leur système. Le premier avantage est que le fonctionnement du Common Layer reste confidentiel. Le second avantage est la simplicité à intégrer le Common Layer. Comme indiqué par la figure 28, peu importe le produit LuxTrust utilisé, les méthodes déclarées dans l'interface ont le même nom et peuvent être appelées de la même manière. De cette façon il n'est pas nécessaire d'écrire de nouvelles lignes de code pour chaque méthode d'authentification que l'on souhaite mettre en place.

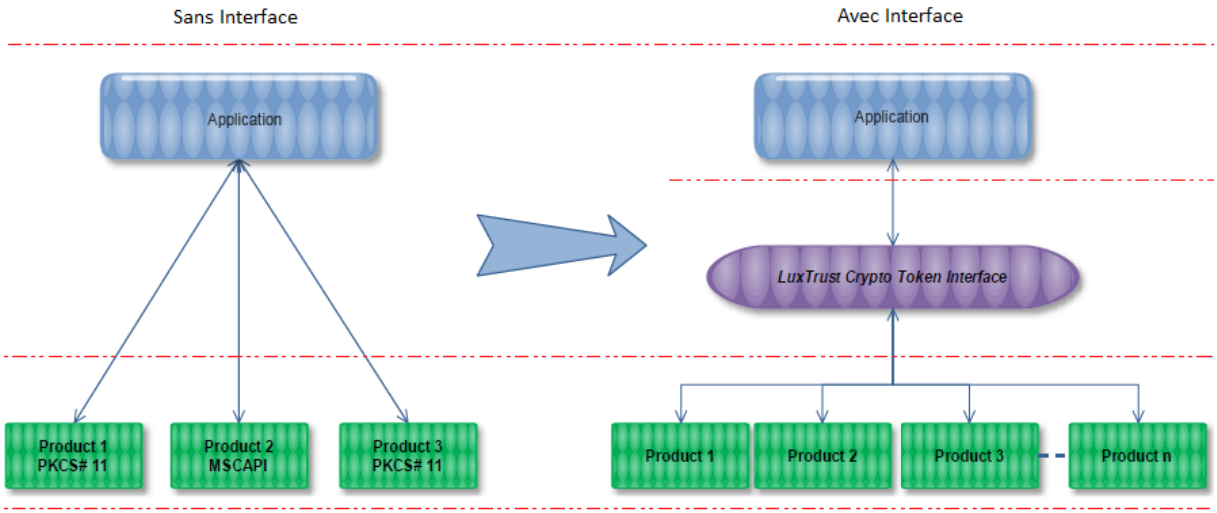

Figure 28 – Comparaison d'une implémentation du Common layer avec et sans interface

L'interface est donc le principal point d'entrée pour les applications. Le client n'a donc pas à connaître les spécifications des serveurs et des protocoles LuxTrust. Il a seulement besoin de s'adapter à cette interface.

Il existe 3 principaux axes d'authentification :

- ce que je sais (login, identifiant, mot de passe)
- ce que je possède (un token, un SigningStick,etc…)
- ce que je suis (empreintes digitales, empreintes rétiniennes, autres paramètres biométrique)

Pour pouvoir proposer des produits facilement utilisables par le plus grand nombre, LuxTrust s'est limité aux deux premiers niveaux de l'authentification. Les certificats sont "possédés" par les utilisateurs et protégés par des mots de passes et des identifiants. Dans un premier temps, une personne qui souhaite commander un certificat va procéder à une commande chez LuxTrust. Pour finaliser cette commande, LuxTrust impose à ses clients de se rendre physiquement dans des agences partenaires avec une pièce d'identité. Cette procédure garantit qu'un certificat émis pour une personne physique est bien associé à la bonne personne.

4.5.4.2. Analyse du code du Common Layer

La toute première étape est l'analyse du code du Common Layer. Il faut en comprendre le fonctionnement et l'alléger pour le rendre compatible avec la technologie Java Mobile (J2ME) Le Common Layer a été fournie avec le source d'une petite applet de test écrite

en Java. Cette applet utilise des libraires sous forme de jar. Comme nous ne sommes intéressés que par l'authentification par token Vasco, seuls les fichiers suivants ont donc été retenus :

- LuxTrust_CommonLayer_3.0.2_Demo_Applet_1.0 : Cette librairie contient les objets et les méthodes nécessaires au fonctionnement de l'application de test.
- LuxTrust_Cryptomathic_VASCO_CryptoTI_Adapter_1.2 : La librairie contient les réimplémentations des objets de BouncyCastle utilisés pour le chiffrement des données.

Il était impossible de modifier des sources et de réintégrer les fichiers compilés dans les librairies jar car chaque fichier est associé à une empreinte qui garantit l'intégrité du contenu du jar. L'objectif était donc de décompiler les fichiers .class des librairies pour les réécrire dans un projet sur lequel j'aurai la main. Pour ce faire deux outils ont été utilisés :

- Jad [3]: C'est un fichier exécutable qui propose tout une série d'options pour interpréter le bytecode contenu dans les .class et proposer un fichier .java.
- Java Decompiler [Dupuy E., 2008] : JD-Eclipse est un plugin qui s'intègre à l'IDE Eclipse. Il permet de visualiser les .class comme des sources Java.

Il est nécessaire d'utiliser au moins deux outils distincts car un décompilateur interprète du langage machine (le Bytecode) et le résultat est souvent incohérent. Il faut donc analyser tous les codes sources générés et les retravailler pour qu'ils soient cohérents. Une fois que l'application capable de réagir comme l'application de test LuxTrustavait été réalisée, il est devenu possible de modifier certains sources pour prendre en compte un fichier .properties de configuration personnalisé et ainsi aller contacter le serveur de test LuxTrust à la place du serveur de Production. Pour ces tests, LuxTrust a fourni un token ainsi que les login et mot de passe associé. Pas à pas il a fallu sortir des fichiers .class des librairies pour les intégrer à au projet local, toujours en testant régulièrement la compatibilité de du travail réalisé avec le serveur de test LuxTrust. Il était très important de pouvoir garantir l'intégrité de la solution portée. Ce travail a été réalisé à partir de l'IDE Eclipse [Foundation, 2012] sur une machine équipée de la version 6 de Java. A la fin de cette étape, la version de l'application de test utilisée pour mon analyse était affranchie des jars Cryptomatic. Il est ainsi devenu possible d'approfondir l'étude du fonctionnement du Common Layer. Dans un premier temps, seuls les logins et mots de

[3] Copyright 2001 Pavel Kouznetsov

passe sont transmis. Si ces deux informations sont validées, l'utilisateur reçoit un certificat contenant la clef publique de LuxTrust. L'OTP généré par le token Vasco va être chiffré en RSA avec cette clef et sera retransmis au serveur d'authentification. C'est seulement si l'OTP reçu correspond à l'OTP attendu que LuxTrust transmet son certificat à l'utilisateur. Une fois cette étape réalisée, il est devenu possible de porter ce code sur une plateforme de type Android.

4.5.5. Portage du Common Layer avec la technologie J2ME

Pour développer une application Android, il a fallu installer le JDK SDK Android [Guignard D., 2011]. Un SDK est un kit de développement regroupant un ensemble d'outils permettant aux développeurs de créer des applications de type défini et d'exploiter toutes les fonctionnalités du téléphone telles que le GPS, l'appareil photo ou la mémoire interne. Il est possible de l'installer en tant que module externe d'Eclipse. De cette façon, le SDK est complètement intégré à l'IDE La version téléchargée est la 1.6. J'ai commencé par développer une interface graphique simple (figure 29) pour me familiariser avec le développement Android.

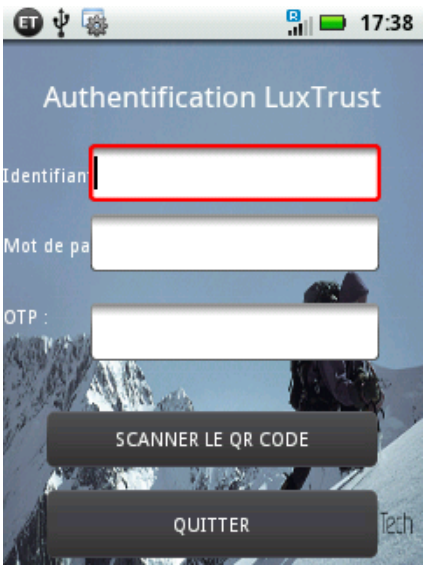

Figure 29 - Capture d'écran de l'application Android

L'utilisateur va pouvoir saisir son identifiant et son mot de passe LuxTrust puis saisir l'OTP généré par son Token Vasco. Les deux boutons d'action vont soit mettre fin à l'application, soit démarrer le processus. Une méthode de contrôle simple va permettre de ne lancer le scan du code que si toutes les informations ont été saisies.

Pour l'utilisation de l'appareil photo, le scan du QR Code et son interprétation, j'ai choisi de ne pas réimplémenter toute la fonctionnalité. J'ai préféré utiliser ZXing [Owen S., 2007] (qui se prononce "zebra crossing") qui est une librairie open-source, multi-format (et justement sous Android) qui permet le traitement de l'image de code à barres ou de QR Code. Elle permet d'exploiter la caméra intégrée sur les téléphones mobiles pour lire et décoder des codes à barres sur l'appareil, sans communiquer avec un serveur. L'utilisation de cette librairie est autorisée dans le cadre d'un usage non commercial. Lorsque le projet sera industrialisé, il faudra réécrire les fonctions de scan et de traitement de l'image pour s'affranchir des librairies ZXing. Le QR Code est interprété pour récupérer le jeton que LuxTrust a transmis au serveur Web. A ce niveau de l'utilisation, l'application a les paramètres d'authentification de l'utilisateur et le jeton scanné sur le site internet. J'ai créé une classe ContactLuxTrust.java qui s'enrichit avec ces informations et qui fait appel à toutes les classes qui implémentent le Common Layer.

Réutiliser le code récupéré en décodant les fichiers Jar n'est pas suffisant car les deux versions de Java sont différentes et sur certains points, la version J2ME réagit différemment de la version standard. Par exemple la classe *org.w3c.dom.Node* n'est pas implémentée de la même façon pour les deux versions de Java. L'objet a évolué et deux nouveaux attributs ont été ajoutés, baseURI et textContent. La méthode getTextContent permet de retrouver ce second paramètre. Toutefois les versions d'Android 2.1 et 2.3 utilisées dans le cadre de mon projet sont antérieures à cette évolution. Des recherches sur internet m'ont permis de remplacer ces appels par getFirstChild().getNodeValue(). Mais la plus grosse difficulté rencontrée reste l'appel aux outils de chiffrement. Les jar de LuxTrust font appel aux classes de la librairie BouncyCastle [Castle T. L., 2002]. Or cette librairie ne tourne pas sous Android. Dans la version standard de Java, il est possible d'ajouter des constructeurs vides dans les classes. La JVM se charge d'initialiser les attributs lors de l'instanciation de l'objet. Ce n'est pas le cas de la version Micro de Java qui, si elle tolère des constructeurs vides à la compilation, va avoir un fonctionnement différent à l'utilisation. Les attributs ne sont pas initialisés et on retrouve des valeurs à null qui vont fausser les tests logiques voire même générer des exceptions de type NullPointerException. La seule solution est de réécrire les classes de BouncyCastle pour réimplémenter les constructeurs et forcer les initialisations. Avant de me lancer dans cette tâche ardue, j'ai effectué des recherches sur différents sites et forums traitant du développement sous Android. J'ai fini par trouver le site d'une personne qui avait rencontré la même situation et qui avait pris le temps de porter la BouncyCastle sur Android. Son projet se nomme SpongyCastle [Tyley R., 2012]. En intégrant cette librairie

dans mon application, le chiffrement et la gestion des certificats sont devenus possibles depuis un Smartphone de type Android. L'application Java pour Android se compose de 207 classes Java mais on peut en retenir 6 qui composent le cœur de l'application :

- AndroidAppActivity.java est la classe principale qui va gérer l'application et son fonctionnement. Elle va exécuter les différentes étapes (contrôle des saisies, scan du QR Code, appel à la Common Layer, affichage du résultat de l'authentification) et traiter les codes retours de chacune d'elles.
- Controles.java va simplement contrôler la saisie de l'utilisateur. Elle retourne un code 0, 1, 2 ou 3, 0 étant le cas où rien ne manque.
- DisplayResult.java est exécuté après l'appel au Common Layer. L'utilisateur est ainsi informé du résultat de son authentification sur le Smartphone. Dans le cas d'une erreur, l'utilisateur saura s'il cela est dû à une erreur de saisie, à un problème technique au niveau de l'application ou à une erreur de communication avec les serveurs LuxTrust.
- IntentIntegrator.java et IntentResult.java sont les classes qui utilisent ZXing. Elles vont utiliser l'objectif de la caméra pour scanner un QR Code et interpréter le résultat qui sera retourné sous forme de chaine de caractère.
- ContactLuxTrust.java fait appel à la Common Layer pour communiquer avec les serveurs LuxTrust de façon sécurisée.

Ce découpage des classes permet de faciliter la maintenabilité de l'application et la rend plus modulable si une évolution était planifiée.

4.5.6. Le serveur d'authentification

Le serveur d'authentification se composera de trois éléments qui sont une base de données pour stocker les jetons générés et les gérer, un serveur d'application et les webservices que ce dernier va héberger. La structure est décrite par la figure 30.

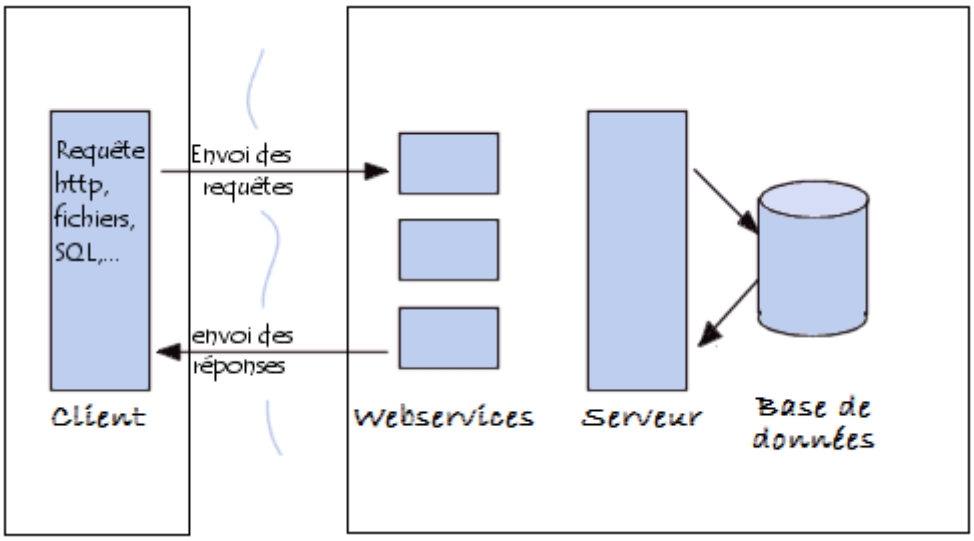

Figure 30 - Structure du serveur d'authentification

Ce module sera, à terme, intégré aux serveurs LuxTrust et doit servir de serveur centralisé pour l'authentification. Il doit avoir deux principales qualités qui sont la fiabilité et la robustesse. En ce sens, nous avions pensé à mettre en place un serveur qui fonctionne à la façon des webservices. Pour rappel, les avantages d'un webservice sont les suivants :

- l'interopérabilité entre différentes plateformes
- l'utilisation de standards et protocoles ouverts
- l'utilisation du protocole http, permettant de contourner les pare-feu
- les échanges peuvent être sécurisés simplement en utilisant SSL, ou de manière plus approfondie avec WS-Security (et autres spécifications de sécurité WS-*).

On pourrait également garantir l'identité de l'utilisateur ainsi que la non-répudiation grâce à un système de signature. Avec l'aide de Ludovic, nous avons mis en place un serveur développé en J2EE et s'exécutant sur un serveur d'application de type Glassfish [Lafosse J., 2011] dans sa version 3.1. Pour la gestion des données nous avons mis en place une base de données MySQL qui contient une table *Record*. Cette table va stocker les demandes de jetons faites par les serveurs web et suivre le cycle de vie des jetons que le serveur va générer. Elle se compose des attributs montrés par la figure 31 :

Champ	Type	Interclassement	Attributs	Null	Défaut	Extra
id	int(11)			Non	*Aucun*	auto_increment
login	char(100)	utf8_bin		Oui	*NULL*	
token	text	utf8_bin		Non	*Aucun*	
tstamp	datetime			Non	*Aucun*	
authenticate	enum('authenticated','non_authenticated','expired')	utf8_bin		Non	non_authenticated	
callback	varchar(250)	utf8_bin		Non	*Aucun*	

Figure 31 – Table des attributs de la base de données du serveur

Le serveur d'authentification va générer un jeton unique à transmettre au serveur web puis va le stocker dans l'attribut "token" avec la date de création dans le champ "tstamp". L'unité du jeton est garantie grâce à la façon dont il est généré. On utilise en effet une fonction de hashage sur l'url de callback concaténée au timestamp correspondant à la demande. Le cycle de vie d'un enregistrement de cette table se définit en 3 étapes. A sa création, l'enregistrement est inséré avec un statut "non_authenticated". Lorsque l'utilisateur valide son authentification via l'application mobile intégrant le Common Layer, le statut de l'enregistrement est mis à jour vers "authenticated". En parallèle, un processus parcourt à intervalle régulier la table Record et compare le timestamp de création avec l'heure courante. Si l'occurrence est toujours en statut "non_authenticated" et qu'il a dépassé la durée de vie définie, le processus change le statut de l'occurrence vers "expired". De cette façon, on gère l'expiration des jetons dans le cas où l'utilisateur ne s'authentifie pas dans le temps imparti. L'expiration et l'unicité des jetons générés sont les principales contraintes que nous avions concernant la réalisation de ce serveur d'authentification. Le reste s'est développé parallèlement à la librairie à intégrer dans les serveurs Web des clients. La première version générait un jeton sur demande puis répondait aux demandes récurrentes des sites clients. Nous avions écrit un webservice *isAuthenticated* qui renvoyait un booléen selon la situation. L'inconvénient de cette solution nous est apparu lors d'une réflexion sur l'intensification de l'utilisation de notre solution. Si plusieurs serveurs reviennent à intervalles réguliers demander à LuxTrust si leur jeton est validé ou périmé, le trafic sur le réseau va vite saturer. Nous avons donc pensé à un nouveau principe utilisant les URL de callback. Lors d'une demande de jeton, le serveur web va transmettre une adresse internet à LuxTrust et se mettre à l'écoute à la façon d'un webservice. Suite à ceci, aucune communication n'est réalisée tant que l'authentification n'a pas été validée par le Smartphone de l'utilisateur. L'expiration du jeton est aussi bien gérée du côté LuxTrust que du côté Serveur web. De ce fait on a un trafic qui se limite au strict minimum.

4.5.7. Le site client

Maintenant que nous avons un serveur capable de générer les jetons puis de valider l'authentification, il faut développer la partie qui s'intègrera dans les serveurs web qui gèrent les sites internet clients. La solution mise en place se devait d'être conviviale et efficace. Il est important que le fonctionnement soit suffisamment simple pour ne pas rebuter l'utilisateur et suffisamment évolué techniquement pour ne pas surcharger le réseau avec un trafic inutile.

4.5.7.1. Le premier prototype

L'utilisateur va visiter un site internet sur lequel il va initier une demande de connexion. Le serveur web qui gère le site va contacter le serveur d'authentification via notre webservice pour demander un jeton de connexion. Pour ce faire il va transmettre un identifiant de session ainsi qu'une URL de callback. Le serveur d'authentification va générer un jeton unique et le transmettre en réponse à l'appel. Le serveur web convertit alors ce jeton en QR Code qui est envoyé pour être affiché sur le navigateur internet de l'utilisateur.

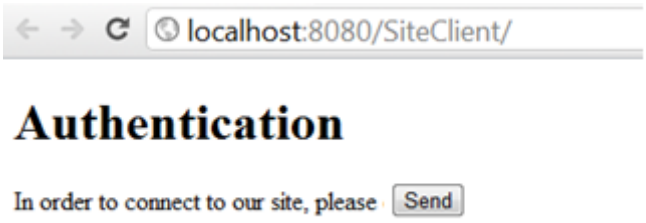

Figure 32 – Page d'accueil du premier prototype

La première version du prototype (figure 32) a été développée en Java en J2EE. Les communications entre le serveur web et le serveur d'authentification se faisaient via des webservices de type SOAP.

Figure 33 – QR Code affiché par le premier prototype

L'utilisateur affiche la page du site client sur son navigateur favori. Il est ensuite invité à cliquer sur le bouton *Send*. Ce bouton va lancer le processus en demandant la génération d'un jeton au serveur d'authentification. Lorsque le jeton est reçu, il est affiché sous forme d'un QR Code (figure 33) pour être scanné par l'application mobile. La question que nous nous sommes posée concernait le rafraichissement de la page dans le navigateur internet du client. Dans un premier temps, nous avons ajouté un lien que l'utilisateur devait cliquer pour forcer le rafraichissement de la page et lancer l'appel au webservice qui vérifie l'authentification.

Figure 34 - Premier prototype réalisé : authentification en erreur

Dans le cas où l'authentification n'est pas validée, l'utilisateur est informé du retour du webservice (figure 34) et doit relancer le rafraichissement plus tard. Dans le cas contraire, l'utilisateur est informé que la connexion est validée. L'accès à des pages personnalisées voire protégées est maintenant possible (figure 35).

Figure 35 - Premier prototype réalisé : authentification réussie

Si cette solution est très simple à implémenter, son principal défaut est la nécessité de cliquer sur un lien pour forcer le rafraichissement. En suivant cet axe de réflexion, il est tout à fait possible d'optimiser le côté convivial en automatisant le rafraichissement de la page internet. On peut également analyser le trafic entre les différents éléments de l'architecture. Ce prototype nécessite une action de l'utilisateur pour forcer le rafraichissement. Mais l'utilisateur impatient va pouvoir forcer le rafraichissement de sa page internet autant de fois qu'il le voudra. Cela signifie que des données transiteront entre l'ordinateur de l'utilisateur et le serveur web gérant le site client. A chaque requête, le serveur web lance un appel au webservice isAuthenticated vers le serveur d'authentification. Ajoutons à cela les différents retours pour chacune de ces requêtes, le trafic peut saturer très rapidement (figure 41).

4.5.7.2. Le deuxième prototype

La deuxième version a été améliorée afin d'optimiser la convivialité. Le bouton "Send" a été supprimé de la page car nous souhaitions automatiser le rafraichissement de la page internet. Ceci a été réalisé avec une balise html insérée dans le code de la page (figure 36)

```
<meta http-equiv="refresh" content="5">
```

Figure 36 - Balise HTML pour le rafraichissement automatique

Le QR Code est toujours affiché mais nous n'avons plus de bouton. La page se rafraichit d'elle-même toutes les 5 secondes ce qui déclenche l'appel au Webservice isAuthenticated (figure 37).

Figure 37 – QR Code affiché par le second prototype

L'utilisateur n'est pas informé à chaque rafraichissement de la non-validation de son authentification. La page internet du navigateur va changer seulement lorsque le webservice isAuthenticated retourne la valeur "*true*" ou "*expired*". Si cette solution est plus conviviale que la précédente, elle offre une présentation nettement dégradée. En effet le résultat de la balise HTML *Resfresh* est un véritable rechargement de la page internet. Toutes les 5 secondes, la fenêtre du navigateur internet devient blanche et se recharge depuis le serveur web qui gère le site client. On note également que le problème de saturation du réseau que nous avions noté avec le premier prototype n'est pas résolu. Les communications entre les différents éléments restent très nombreuses, fréquentes et inadaptées à un déploiement de la solution à l'échelle industrielle (figure 41) car même si l'utilisateur n'est plus sollicité, le rafraichissement est fixé avec une fréquence de 5 secondes.

4.5.7.3. Le troisième prototype

Pour le troisième prototype, l'objectif était de conserver l'aspect de rafraichissement automatique pour ne pas impliquer l'utilisateur dans cette étape. Il est donc devenu incontournable de passer vers une autre technologie qui offrirait des composants plus

évolués. Il existe un framework Java J2EE appelé JSF (Java Server Faces). Ce framework nous permet d'utiliser une bibliothèque de composants appelée PrimeFaces. Tout d'abord, les composants graphiques à disposition permettent de réaliser de façon très simple des sites internet plus esthétiques comme le montre la figure 38.

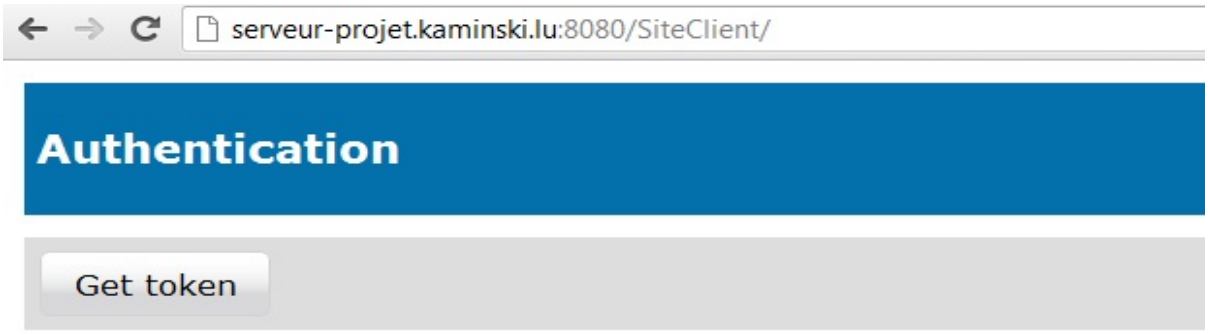

Figure 38 - Troisième prototype réalisé

Lorsque l'utilisateur va cliquer sur le bouton "Get token", la demande d'authentification est réalisée vers le serveur d'authentification et le site affiche le token qu'il reçoit sous forme de QR Code (figure 39).

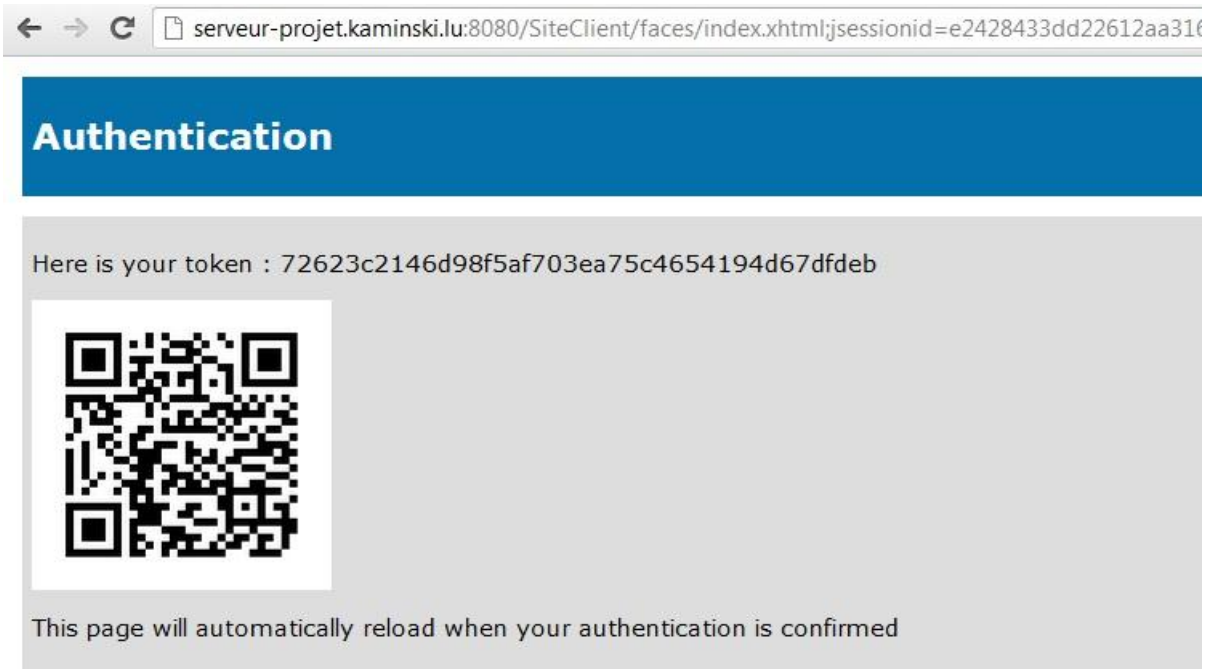

Figure 39 - Troisième prototype réalisé

L'utilisation de la librairie PrimeFaces nous a permis d'utiliser des composants appelés Ajax. Ces derniers révolutionnent le fonctionnement de notre site client. Dans un cas classique, ce qui concerne nos deux premiers prototypes, le site internet effectue du "*pull*". Ce terme, qui en anglais signifie "tirer", illustre le fonctionnement d'un site qui va

aller chercher une information sur son serveur web. Grâce aux composants Ajax associés à du code Javascript, on peut maintenant implémenter un fonctionnement en mode "push". Ce terme, qui signifie "pousser" illustre un mode de fonctionnement dans lequel le serveur web va "pousser" des informations vers le site internet. Ainsi, lorsque le serveur web reçoit le signal du serveur d'authentification indiquant que l'authentification est réussie, il envoie un signal au composant Ajax qui va forcer le rafraichissement de la page internet dans le navigateur de l'utilisateur et lui donner accès aux pages protégées (figure 40).

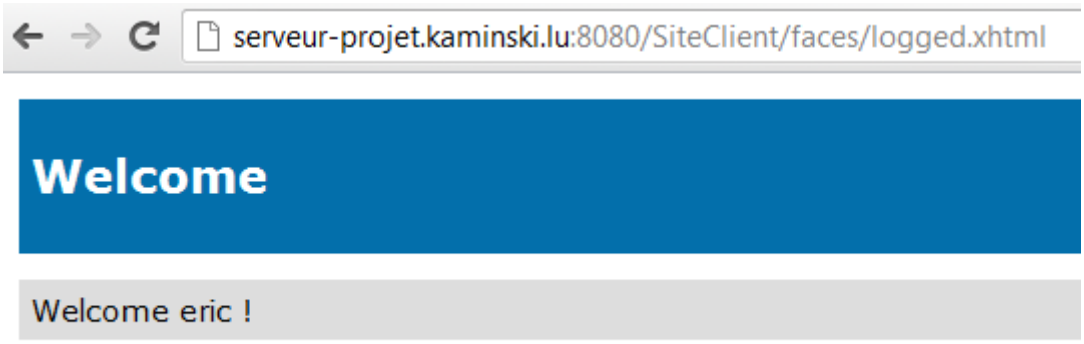

Figure 40 - Résultat suite à une authentification validée

En analysant le trafic entre les différents éléments de l'architecture (figure 41), on note qu'on a réussi à diminuer nettement le trafic entre l'ordinateur de l'utilisateur et le serveur web qui gère le site client. Par contre, on garde un trafic très lourd entre ce dernier et le serveur d'authentification. Il faut donc faire évoluer le type de technologie utilisée entre ces deux éléments ou changer le protocole de communication.

4.5.7.4. Le quatrième prototype

En imaginant un déploiement à grande échelle et une intégration du serveur d'authentification aux systèmes LuxTrust, il faut être conscient que LuxTrust ne pourrait pas gérer des connexions récurrentes de plusieurs dizaines de serveurs multipliées par des dizaines d'utilisateurs qui demanderaient des connexions simultanément. Le troisième prototype nous a permis d'arriver à une solution qui offre une interface esthétique et conviviale et un trafic peu chargé entre le serveur web et l'ordinateur de

l'utilisateur. Dans cette version, le serveur web doit encore contacter le serveur d'authentification régulièrement pour vérifier le statut de l'authentification ("validée", "en attente" ou "expirée"). Je me suis orienté vers la mise en place d'un mécanisme utilisant des URL de Callback. Lorsqu'il demande un jeton au serveur d'authentification, le serveur web va également transmettre une adresse internet vers laquelle le serveur d'authentification pourra envoyer un signal lorsque l'authentification sera validée. Lorsque le jeton est reçu, il est transmis vers le navigateur internet de l'utilisateur pour être affiché sous forme de QR Code puis le serveur web se met à l'écoute à la façon d'un webservice.

A la réception du signal, le serveur web contacte le composant Ajax pour demander le rafraichissement de la page internet. La figure 41 illustre l'objectif que nous avons atteint et met en valeur les avantages par rapports aux précédents prototypes. Le trafic est optimisé entre tous les éléments de l'architecture. Les communications entre les différents serveurs sont limitées au strict minimum et ne transitent que lorsqu'une action est à réaliser.

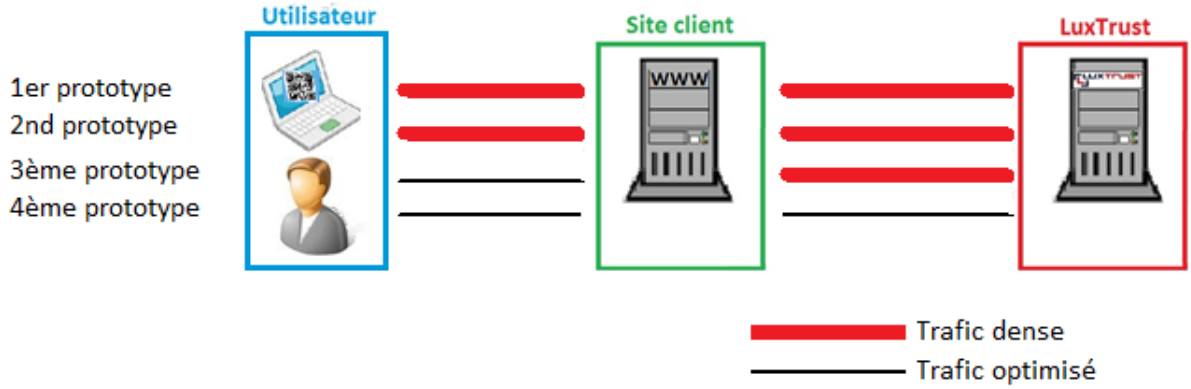

Figure 41 – Evaluation de la densité du trafic entre les différents acteurs

Une économie de trafic est également réalisée grâce au processus qui tourne en parallèle sur le serveur web et sur le serveur d'authentification. Ce dernier contrôle l'état des jetons et force l'expiration des demandes de jeton si l'authentification n'est pas validée dans le temps imparti sans nécessiter de communication entre les différents éléments de l'architecture.

D'autres technologies auraient pu être utilisées pour faire évoluer la solution. Les composants Ajax permettent de fonctionner en "*polling*" ou en "*long-polling*" mais si nous évitions le rafraichissement supprimé lors du passage au troisième prototype, nous

n'optimisions pas le trafic entre le navigateur internet de l'utilisateur et le serveur web. Nous avions également étudié le fonctionnement des serveurs Comet qui est également utilisable avec un serveur Glassfish en intégrant l'API Grizzly. J'ai choisi de ne pas aller plus loin dans l'évolution de la technologie car le rapport entre la complexité de la solution et les bénéfices devenait insatisfaisant. Il faut également noter que la solution, dans sa quatrième version, respecte le cahier des charges que je m'étais fixé :

- la solution est conviviale et esthétique
- l'unicité des jetons est garantie
- le trafic entre les différents éléments de l'architecture n'est pas saturé
- l'expiration des jetons est gérée sur le serveur web et le serveur LuxTrust
- l'architecture est souple. Sans programmer de grosses modifications, elle peut être adaptée à des sites internet différents mais également à un système de sécurité gérant l'accès à un bâtiment

4.5.8. Réalisation d'un prototype pour iPhone

A ce stade du projet, nous sommes arrivés à mettre en place une solution complète et fonctionnelle d'authentification à partir d'un Smartphone de type Android. L'analyse du marché avait montré que le concurrent direct de Google était Apple avec ses appareils de type iPhone. Nous avons donc décidé de porter notre prototype sur iOS [Napier R. et Kumar M., 2012] [Allen S., Graupera V. et Lundrigan L., 2010]. Les applications faites pour les appareils Apple ne sont pas écrite en Java, dans un premier temps, il faudrait donc porter le code depuis Java vers Objective-C. La question qui s'est posé lors du portage Android pourrait se poser également ici, qu'en est-il de l'utilisation de la librairie de chiffrement ? La Bouncy Castle est disponible en C, langage compatible avec l'Objective-C. Le temps passé sur l'ensemble du projet a été conséquent. J'ai préféré analyser la faisabilité du prototype iOS et relever les difficultés que je pourrais rencontrer plutôt que de réécrire du code d'un langage vers un autre une seconde fois. Ma société a mis à ma disposition un Macbook sur lequel était installé l'IDE X-Code. Ce dernier permet de compiler du code Objective-C mais également de créer des interfaces graphiques et de déployer ses projets sur un émulateur ou sur un véritable iPhone. La gestion de l'interface graphique et l'arborescence du projet est assez similaire à ce que je rencontrais sous Eclipse. En revanche il m'a été difficile de m'habituer au langage de programmation. En plus de la syntaxe, la logique de programmation est totalement différente. Par exemple, un héritage du C est la gestion de la mémoire qu'il faut réaliser

manuellement lors de l'allocation de zone mémoire pour de nouveaux objets. Mais le plus perturbant a sans doute été la gestion des méthodes et leurs appels. En C ou en Java, une méthode est définie par un nom et le nombre de ses paramètres et on l'appelle par son nom en passant les bons paramètres. En Objective-C, l'entête d'une méthode va être composé par un nom et chaque paramètre sera défini par un nom, un type et une troisième variable. J'ai eu du mal à développer une interface graphique pourtant simple. Afin de ne pas déborder sur la durée de mon projet, je me suis limité à une application qui scanne le QR Code et qui contacte le même webservice que la libraire installée sur les serveurs LuxTrust. De cette façon on a un prototype qui permet la connexion à un site internet affichant un QRCode sans devoir implémenter tout le Common Layer.

Mais même cette dernière étape, l'appel au Webservice, a été compliqué sur iOS. Là où Java propose une API riche avec laquelle sont passés les paramètres, le type d'appel et les paramètres, iOS nous impose de réécrire nos objets. Des recherches sur internet m'ont permis d'écrire une classe ServerAPI.m et son interface ServerAPI.h dont les sources se trouvent en annexe de ce mémoire. Le fonctionnement de l'application iPhone (figure 42) est similaire à celui de l'application Android. A ceci près que la saisie d'un OTP n'est pas nécessaire car nous ne contactons pas les serveurs LuxTrust. L'utilisateur saisit seulement son identifiant et son mot de passe puis clique sur le bouton pour scanner le QR Code visible sur le site internet.

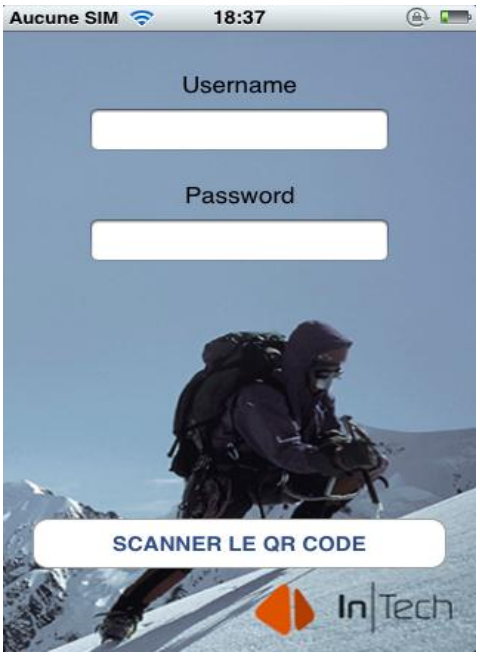

Figure 42 - Capture d'écran de l'application iPhone

L'application n'implémente pas le Common Layer, elle se contente de contacter le Webservice comme le ferait LuxTrust. En ce sens, L'URL de Callback a été écrite en dur dans le code. L'objectif ce prototype était de montrer qu'il était possible d'implémenter l'appel à un webservice dans un environnement iOS et d'utiliser une librairie C. L'utilisation d'une librairie C standard en Objective-C se réalise sans difficulté. On peut donc se rassurer sur l'utilisation de la BouncyCastle et de ses outils dans une application iPhone. L'intégration de la Common Layer sur cette plateforme pourra donc être planifiée ultérieurement pour ne pas faire déborder le planning de mon projet.

5. Déploiement, évaluation et tests

5.1. Déploiement

5.1.1. L'application Smartphone

Les premières versions de la solution ont été réalisées, déployées et testées sur l'ordinateur de Ludovic qui hébergeait un serveur Glassfish. La machine hébergeait également un serveur Wamp qui lui permettait de gérer une base de données mySQL ainsi qu'un serveur web pour les pages HTML du site client. A l'aide d'une application basique tournant sous Android, Ludovic réussissait à scanner le QR Code affiché par son site client et transmettait un signal vers son serveur d'authentification. Cette dernière action avait pour résultat de rafraichir le site client pour afficher une page personnalisée. Très rapidement j'ai souhaité dépasser le cadre des tests en local sur une même machine où tous les serveurs cohabitent. Plutôt que de déployer nos serveurs sur le réseau local d'InTech, nous avons mis en place un serveur d'application GlassFish sur un ordinateur raccordé à internet en permanence et situé en France. Sur un deuxième ordinateur, nous avons installé un serveur Wamp afin de migrer la base de données mySQL et le serveur web hébergeant le site client.

A partir de ce point, les tests que nous réalisions dépassaient même le cadre du réseau local et nous avions des retours déjà plus probants. La seule limite que nous avons rencontrée concerne l'intégration de notre serveur d'authentification à l'environnement LuxTrust. Du fait d'une politique de sécurité très restrictive, il ne nous a pas été possible d'intégrer un module à leur environnement de test. Le schéma que nous avons réalisé est alors le suivant :

- nous affichions la page internet du site client
- le serveur web contactait le serveur d'authentification pour demander le jeton de connexion à afficher sous forme de QR Code.
- nous scannions ce QR Code à l'aide l'application Android intégrant le Common Layer.
- l'application Android contactait les serveurs de LuxTrust via le Common Layer intégré.

- si le retour du Common Layer était positif, l'application Android affichait un feedback puis contactait un webservice à l'écoute sur notre serveur d'authentification.
- le serveur d'authentification envoyait alors le signal attendu par le serveur web pour lancer le rafraichissement de la page internet sur notre navigateur.

Bien évidemment, notre serveur d'authentification doit être à terme intégré aux serveurs de LuxTrust. Ces derniers, une fois que l'utilisateur s'est authentifié, se chargent d'envoyer le signal vers le serveur web qui gère le site client. La possibilité pour une application Smartphone d'appeler un webservice à l'écoute sur le serveur d'authentification nous a permis de réaliser la version iPhone sans trop de bouleversement. Cette dernière connait deux paires d'identifiants de connexion qui sont codées en dur. Si les données saisies correspondent aux données attendues, l'application réalise simplement un appel au webservice pour que le serveur d'authentification se charge de commander le rafraichissement au serveur web. Rappelons ici que la version iPhone n'est qu'un prototype sur lequel le Common Layer n'a pas été implémenté. Il était donc exclu de le faire contacter les serveurs LuxTrust. Les tests Smartphone ont été réalisés sur un téléphone de type iPhone 3GS 32GB. La version Android, quant à elle, a été déployée sur deux téléphones Android 2.1 (Eclair) et 2.3 (Gingerbread). Les communications vers les serveurs LuxTrust se faisaient soit par un réseau Wifi, soit par le réseau 3G.

5.1.2. Le serveur d'authentification

Concernant la configuration de serveur d'application [Lafosse J., 2011], il a été nécessaire de réaliser quelques personnalisations à travers la console d'administration. Il a fallu également ajouter un module pour le rendre pleinement fonctionnel. Dans sa version de base, Glassfish n'est pas disposé à communiquer avec un serveur de base de données de type MySQL. Il a fallu récupérer le driver adéquat sur le site d'Oracle (mysql-connector-java-5.1.18-bin.jar) dans le répertoire d'installation de Glassfish. Une fois le driver installé et le serveur redémarré pour en forcer la prise en compte, nous nous sommes attelés à la configuration de la connexion entre le serveur d'authentification et la base de données MySQL.

La première étape consiste à créer le pool de connexion JDBC en procédant comme illustré par la figure 43:

Figure 43 - Déploiement du serveur d'authentification

On remplit les champs nécessaires pour identifier le nouveau pool de connexion (figure 44) puis on passe à la seconde étape.

Figure 44 - Déploiement du serveur d'authentification

A ce stade de la configuration (figure 45) on ajoute les propriétés de la figure 46 :

Figure 45 - Déploiement du serveur d'authentification

Propriété	Valeur
User	ludo
Password	ludo
DatabaseName	authentication_server
Url	jdbc:mysql://serveur-projet.kaminski.lu:3306/authentication_server
ServerName	Localhost
PortNumber	3306

Figure 46 - Propriétés à ajouter pour la configuration du pool de connexion

A ce stade, le serveur Glassfish est maintenant configuré pour utiliser la base de données MySQL créée sur la machine joignable à l'adresse serveur-projet.kaminski.lu. Concernant les fichiers sources utilisés pour la configuration du serveur ils sont au nombre de trois :

- Glassfish-resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <jdbc-resource enabled="true" jndi-name="jdbc/authentication_server" object-
type="user" pool-name="mysql">
    <description/>
  </jdbc-resource>
  <jdbc-connection-pool allow-non-component-callers="false" associate-with-
thread="false" connection-creation-retry-attempts="0" connection-creation-retry-
interval-in-seconds="10" connection-leak-reclaim="false" connection-leak-timeout-
in-seconds="0" connection-validation-method="auto-commit" datasource-
classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" fail-all-
connections="false" idle-timeout-in-seconds="300" is-connection-validation-
required="false" is-isolation-level-guaranteed="true" lazy-connection-
association="false" lazy-connection-enlistment="false" match-connections="false"
max-connection-usage-count="0" max-pool-size="32" max-wait-time-in-millis="60000"
name="mysql" non-transactional-connections="false" ping="false" pool-resize-
quantity="2" pooling="true" res-type="javax.sql.DataSource" statement-cache-
size="0" statement-leak-reclaim="false" statement-leak-timeout-in-seconds="0"
statement-timeout-in-seconds="-1" steady-pool-size="8" validate-atmost-once-
period-in-seconds="0" wrap-jdbc-objects="false">
    <property name="URL" value="jdbc:mysql://serveur-
projet.kaminski.lu:3306/authentication_server"/>
    <property name="User" value="ludo"/>
    <property name="Password" value="ludo"/>
  </jdbc-connection-pool>
</resources>
```

- Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <resource-ref>
        <res-ref-name>jdbc/authentication_server</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
        <res-sharing-scope>Shareable</res-sharing-scope>
    </resource-ref>
</web-app>
```

- Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="AuthenticationServerPU" transaction-type="JTA">
    <jta-data-source>jdbc/authentication_server</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties/>
  </persistence-unit>
</persistence>
```

Le serveur étant configuré, il ne reste qu'à déployer notre serveur d'authentification. Il faut commencer par exporter le code au format .war. L'IDE Eclipse que nous utilisons permet l'exportation très simplement. Le fichier obtenu est téléchargé sur le serveur d'application à travers la console d'administration (figure 47) :

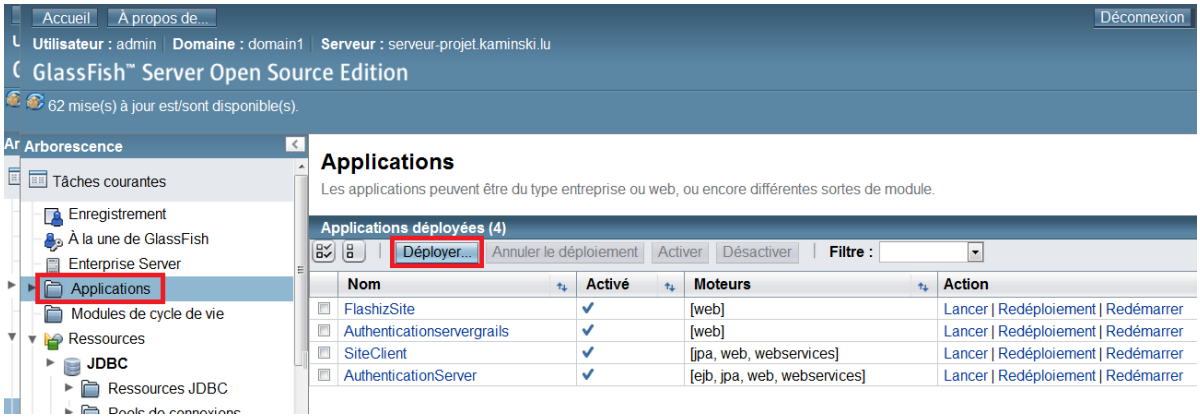

Figure 47 - Déploiement du serveur d'authentification

Le fichier descriptif du web service est donc maintenant accessible à l'adresse http://serveur-projet.kaminski.lu:8080/Authentication/Authentication?WSDL

Il est en annexe de ce mémoire.

5.1.3. Le site client

La configuration de la dernière version du serveur web s'est déployée de la même façon que celle du serveur d'authentification. Les différences vont être le nom de la base de données à contacter. Cette dernière contient une table *user* avec les champs suivant :

	Champ	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
☐	**id**	int(11)			Non	*Aucun*	auto_increment	
☐	**login**	varchar(100)	utf8_bin		Oui	*NULL*		
☐	**token**	text	utf8_bin		Non	*Aucun*		

Figure 48 – Tableau des champs de la table USER

5.1.4. Déploiement de la solution chez Mobey S.A.

La dernière étape du projet a été planifiée une fois le prototype final validé. Il nous a fallu nous plonger dans l'architecture du site de FlashIz pour comprendre la façon dont il a été développé. Alors que nos prototypes étaient réalisés en Java avec le framework J2EE, l'objectif que nous devions atteindre cette fois était un prototype réalisé en Groovy avec le framework Grails. Groovy est un langage léger récemment conçu qui permet de développer des applications qui peuvent s'exécuter avec une Machine Virtuelle Java. Il nous a fallu revoir le rafraichissement entre le serveur web et le poste de travail de l'utilisateur car la librairie PrimeFaces et ses composants ne sont pas disponibles avec Grails. Après avoir étudié et comparé les différents composants Grails natifs, nous avons préféré réécrire les outils AJAX en Javascript. Le travail à fournir a donc été un peu plus compliqué que prévu mais cela nous a permis de conserver la solution que nous avions choisie avec ses avantages sur les économies de trafic. La figure 49 représente le prototype de nouveau site de FlashIz. Le bouton "Connexion" fait maintenant apparaitre un QR Code qu'il faut scanner avec l'application Smartphone FlashIz.

Figure 49 - Intégration au site de FlashIz

5.2. Les tests de validation de l'application

Les tests se sont faits sur deux fronts indépendants :

- Ludovic a développé une petite application déployée sur un serveur d'application. Ce serveur d'application gère une base de données contenant des Logins et des mots de passe. Il a ensuite développé un site web avec un bouton de connexion. Lorsque ce bouton est pressé, le site web envoie une requête au Serveur d'application, reçoit un jeton et l'affiche sous forme de QR Code. Il a ensuite développé une petite application basique sous Android qui scanne ce QR Code et fait appel à un webservice hébergé par le Serveur d'application en transmettant un login et un mot de passe connus. Le serveur d'application contacte alors le serveur web du site client pour lui signifier que l'authentification est réussie.

- De mon côté je réalisais le portage du Common Layer sur le système d'exploitation mobile Android. LuxTrust a mis à ma disposition un Token contenant un certificat enregistré sur leur environnement de test ainsi que l'adresse IP de ce serveur. J'avais donc un login, un mot de passe et un générateur d'OTP pour tenter de me connecter au serveur de test basé chez LuxTrust. LuxTrust n'a pas encore intégré la librairie de Ludovic sur son serveur d'authentification. Afin de pouvoir tester et faire avancer le projet, mon application Android va effectuer l'authentification chez LuxTrust, et si la réponse

est positive, elle fait appel au webservice du serveur d'application mis en place par Ludovic pour transmettre une réponse vers l'url de callback du site web client.

5.3. Les tests unitaires

Afin de compléter l'évaluation de ma solution, j'ai également réalisé des tests unitaires [Gantaume B., 2011] sur les classes Java de l'application mobile. Pour cela, j'ai utilisé le framework JUnit Android disponible sous Eclipse. Cet outil permet de créer des méthodes de test pour chacune des méthodes du projet. Tout d'abord, il faut réfléchir aux différents cas qui peuvent se présenter. Lorsqu'on a une idée complète de ces différents cas, on appelle la méthode de test qui va réaliser plusieurs initialisations. Prenons le cas de la méthode Controles.testCheckFields().

```
public int checkFields() {
	Log.d(Controles.TAG,"checkFields");
	if (login == null || login.trim() == "" || login.length() == 0)
		return 1;

	if (password ==null || password.trim() == "" || password.length() == 0)
		return 2;

	if (OTP == null || OTP.trim() == "" || OTP.length() == 0 )
		return 3;

	return 0;
}
```

Cette méthode vérifie que la saisie a été faite par l'utilisateur. Si ce dernier a oublié d'entrer son login, son mot de passe ou son OTP, un code d'erreur adapté est renvoyé à l'application. La méthode de test réalisée avec l'outil JUnit s'écrit de cette façon:

```
	@Test
	public void testCheckFields() {
		Controles controles = new Controles();
		controles.setLogin(null);
		controles.setPassword(null);
		controles.setOTP(null);
		assertTrue(controles.checkFields() == 1);

		controles.setLogin("test");
		controles.setPassword(null);
```

```
        controles.setOTP(null);
        assertTrue(controles.checkFields() == 2);

        controles.setLogin("Test");
        controles.setPassword("Test");
        controles.setOTP(null);
        assertTrue(controles.checkFields() == 3);

        controles.setLogin("Test");
        controles.setPassword("Test");
        controles.setOTP("Test");
        assertTrue(controles.checkFields() == 0);
        //fail("Not yet implemented");
    }
```

On commence par instancier un objet de la classe à tester puis on initialise les paramètres pour illustrer le cas à tester. La fonction assertTrue(condition) va retourner true ou false en fonction du résultat de la condition. Dans l'exemple ci-dessus, j'ai testé le code retour de la méthode dans un cas où l'utilisateur ne saisit rien. Comme je m'attends à recevoir un code d'erreur à 1, j'écris la condition "`controles.checkFields() == 1`" que je passe en paramètre de assertTrue.

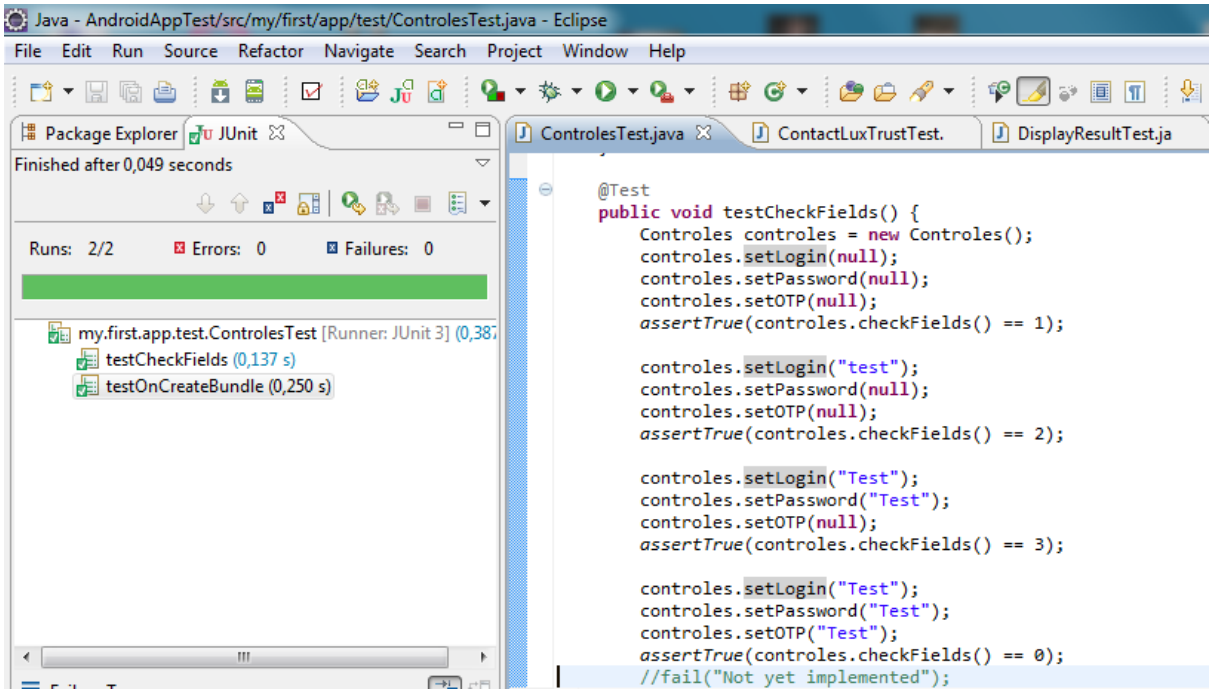

Figure 50 - Résultats de tests JUnit

La figure 50 affiche le résultat des tests JUnit pour la classe ControlesTest.java qui implémente les tests unitaires de la classe Controles.java.

5.4. Bilan des tests

Les tests applicatifs nous ont permis de valider la communication entre les différentes parties de la solution. Le serveur web qui gère le site client est bien capable de réaliser une demande de jeton au serveur d'authentification à travers le réseau Internet et lorsque l'application Smartphone a scanné le QR Code, elle parvient avec succès à contacter l'URL de callback pour donner le signal au serveur Web que l'authentification est réussie et que la page du navigateur internet de l'utilisateur peut être rafraichie.

Le code Java de l'application Android a été validé avec l'écriture de fonction utilisant la librairie JUnit pour Android. En initialisant les différents objets du projet avec des paramètres contrôlés, j'ai pu vérifier que le code renvoyé par chaque méthode correspondait bien aux spécifications fonctionnelles. Lors de l'adaptation de l'application Android Flashiz pour y ajouter la couche d'abstraction LuxTrust, les tests unitaires ont été appliqués aux nouvelles classes pour s'assurer de la non-régression du système.

6. Limites du système

La librairie d'authentification sera à terme intégrée aux systèmes de LuxTrust. Techniquement, l'infrastructure de LuxTrust est préparée aux connexions nombreuses et doit pouvoir parer à un grand nombre de défaillances techniques. La librairie qui s'intègrera au serveur web du site client bénéficiera également des moyens de secours de son hôte. La principale limite du système va se situer du côté de l'application mobile. Lorsque le QR Code est scanné, des données sont envoyées vers le serveur d'authentification basé chez LuxTrust. Il s'agit du jeton, du login, du mot de passe et de l'OTP généré par l'utilisateur. En l'absence de connexion internet (WiFi ou 3G) la transmission sera bien évidemment impossible. Dans le cas où le signal (WiFi ou 3G) est juste suffisant pour valider une connexion internet mais insuffisant pour qu'elle soit stable, l'authentification sur un site internet restera possible. En effet, les données transmises ne sont pas lourde et l'envoi d'un unique paquet est suffisant. Dans le pire des cas, la connexion est perdue et l'application mobile ne reçoit pas le feedback du serveur d'authentification. Mais si les données transmises étaient correctes, le serveur d'authentification aura validé la connexion au serveur web du site client, ce dernier se mettant automatiquement à jour sur l'ordinateur de l'utilisateur.

6.1. Analyser les paquets qui transitent entre le Smartphone et le serveur d'authentification

La communication entre l'application mobile et les serveurs de LuxTrust se déroule comme décrit dans la figure 51 :

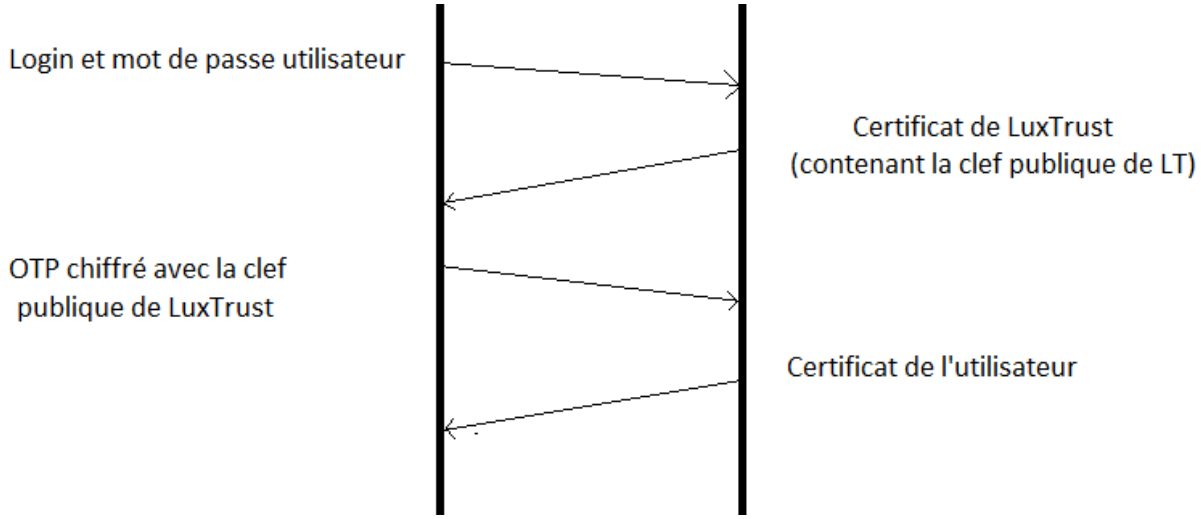

Figure 51 - Fonctionnement du Common Layer

Si le login et le mot de passe de l'utilisateur sont reconnus et validés par LuxTrust, l'application Smartphone va recevoir le certificat de la société LuxTrust avec sa clef publique. L'OTP généré par le token Vasco est chiffré avec la clef publique reçue puis est envoyé au serveur d'authentification. Si l'OTP correspond à une information valide, l'utilisateur reçoit son certificat. Ceci valide l'authentification mais permet également d'exploiter le certificat pour signer, chiffrer ou déchiffrer des données. Ce qu'il apparait lors de l'étude des traces, c'est que les trames échangées ne pèsent pas lourd. Les échanges ne font que quelques bits. L'envoi étant réalisé très vite, il n'est pas donc pas nécessaire de maintenir la connexion au réseau. Le premier paquet qui transite correspond aux identifiants de l'utilisateur. Il s'agit donc de quelques caractères, soit une poignée de bits. LuxTrust va répondre en transmettant son certificat :

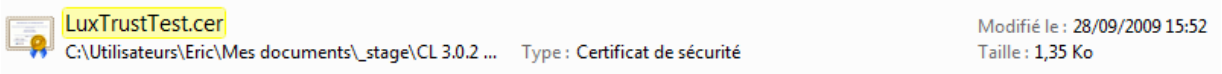

Figure 52 - Certificat LuxTrust sur le File system

Le fichier .cer est une représentation du certificat sur un système de fichier (disque dur, stick USB, etc) Cette représentation (figure 52) pèse 1,35ko. Java n'instancie que les informations nécessaires pour alléger le paquet qui arrivera au destinataire. Ce dernier sera parfaitement capable de reconstituer le .cer à partir des informations qu'il reçoit. Ces informations, en annexe du mémoire, représentent donc des trames de quelques bits. Ensuite, une fois que l'application a reconstitué le .cer, la clef publique est extraite pour chiffrer l'OTP qui est ensuite renvoyé sous forme de hash. D'après les logs de l'application, ce dernier ne pèse que 20 bits. On en déduit donc que même avec un faible

signal, les paquets peuvent être transmis et lancer le processus d'authentification. J'ai volontairement omis de mentionner le retour de LuxTrust car une fois l'OTP transmis, les serveurs d'authentification vont transmettre un signal vers l'URL de callback afin de valider la connexion. Donc même si la couverture réseau ne permet pas à l'application mobile de recevoir le feedback, l'utilisateur verra tout de même le site se mettre à jour avec l'accès protégé devenu disponible.

6.2. Vulnérabilité du système

Une question qu'il faut se poser est la capacité du système à parer des attaques de types "Man in the middle". J'ai étudié différents scénario mettant en scène une personne malveillante qui se placerait à différents endroits de mon architecture. La figure 53 reprend l'architecture de la solution et identifie trois situations de risque :

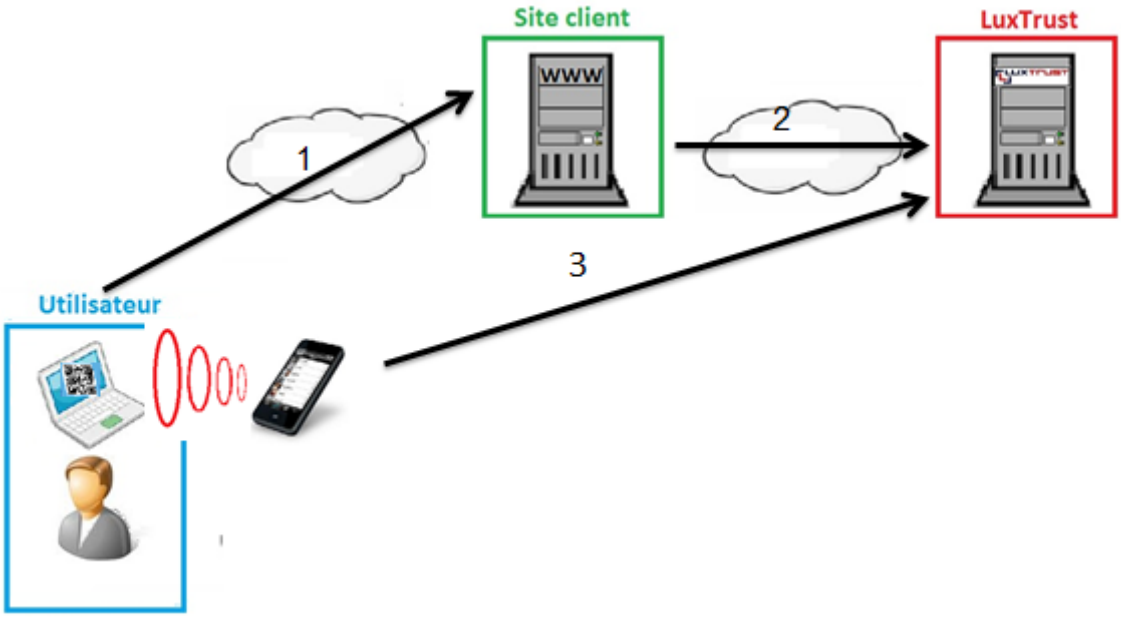

Figure 53 - Schéma de la solution

1. Je suis victime d'une attaque de type "fishing"

Le principe d'une attaque de type "fishing" est qu'une personne malveillante me fait croire que je me trouve sur un site familier ou sur le site que je recherche en recopiant ce dernier. Je pense me connecter sur un site sécurisé, or il s'agit d'une copie destinée à me faire saisir mes identifiants de connexion ou des informations personnelles. Etant

donné que l'utilisateur ne saisit aucun identifiant, Il semble peu probable que le piège soit efficace. Le seul piège qui pourrait être tendu serait un bouton Connexion qui afficherait une QR Code. Mais dans ce cas, même si je scannais ce QR Code, mon application Smartphone recevrait un retour négatif de la part de LuxTrust m'informant que ce QR Code n'est pas un jeton généré par leurs serveurs, ou du moins qu'ils n'ont pas d'URL de callback à contacter. Il est en effet peu probable que le pirate aille en effet jusqu'à faire une demande de jeton à LuxTrust pour être plus crédible. Une attaque de ce type n'aurait donc pas de conséquence. Je souligne ici l'avantage de cette solution qui supprimé la nécessité de saisir des identifiants personnels sur un site internet.

2. Le pirate capte les communications entre le serveur web et le serveur d'authentification

Avec des logiciels adaptés, il est envisageable qu'une personne mal intentionnée puisse "sniffer" les trames qui transitent entre le serveur d'authentification et le serveur web. Pourtant il est probable que l'investissement nécessaire pour une telle attaque ne soit pas amorti. En effet, les communications entre le serveur web et le serveur d'authentification se limitent à de très brefs échanges dont la criticité n'est pas avérée :

- demande de jeton + transmission de l'url de callback
- transmission du jeton sous forme de chaine de caractères
- envoi d'un signal vers l'url de callback

Nous pourrions envisager le cas où la demande de jeton voir la transmission de ce dernier serait bloquée par le pirate. Mais dans ce cas, le site n'afficherait jamais le QR Code. Sans être complètement sûr de la cause de ce problème, l'utilisateur réalisera tout de même que la connexion est impossible. Le pirate pourrait également falsifier l'url de callback. Dans ce cas, le site client afficherait tout de même le QR Code afin que l'utilisateur le scan depuis son Smartphone. L'utilisateur validerait son identité auprès de LuxTrust qui transmettrait son signal vers une fausse URL de callback. Le serveur web ne recevrait donc jamais la validation de LuxTrust et finirait par informer l'utilisateur que son jeton est expiré. Le seul impact d'une attaque de ce type serait donc une interruption de service. En aucun cas, le pirate ne pourrait avoir connaissance de données personnelles concernant l'utilisateur. Si le pirate voulait effectivement piéger un utilisateur il lui faudrait déployer des moyens considérables pour utiliser les deux types d'attaque (le *fishing* et le *Man in the middle*) pour faire croire à l'utilisateur qu'il se

trouve face à un site de confiance, tout en se connectant à un site pour pouvoir capter un QR Code et l'afficher sur son site factice. Enfin, en surveillant les échanges entre le site web auprès duquel il simule une connexion et les serveurs de LuxTrust, il pourrait identifier le signal transmis vers l'URL de callback et rafraîchir son piège. Mais étant donné que l'utilisateur sait qu'il ne sera jamais invité à saisir des informations personnelles, il ne serait pas du tout rentable de déployer autant de moyen. Le transfert d'informations intéressantes se fait réellement entre l'application Smartphone et les serveurs de LuxTrust. C'est ce point qui est encore à étudier.

3. Le pirate arrive à capter les communications passant par le réseau mobile

L'application mobile implémente le Common Layer proposé par LuxTrust pour sécuriser les échanges. Il est déjà compliqué de surveiller le réseau mobile (GPRS ou 3G) pour en analyser les trames. Dans le cas de connexions de type "data", il est possible de chiffrer les communications via un protocole HTTPS ce qui complique encore la tâche d'une personne malveillante. Dans notre cas, en implémentant le Common Layer sur l'application mobile, les échanges sont vraiment chiffrés et transmis via un protocole propriétaire. Le pirate pourrait éventuellement comprendre le protocole de communication implémenté par le Common Layer, cela lui permettrait de déchiffrer le login et le mot de passe de l'utilisateur. Lorsque cette information est reçue par LuxTrust, l'utilisateur reçoit la clef publique de l'Autorité de Certification pour pouvoir chiffrer l'OTP qu'il génère avec son token Vasco. Afin de pouvoir récupérer le certificat et la signature électronique de l'utilisateur, le pirate devrait posséder l'exacte copie du token de l'utilisateur pour pouvoir générer le même OTP au même moment. Il est donc difficilement concevable que les informations sensibles comme les identifiants et le certificat soient récupérés par un tiers malveillant. Le système n'a pas la prétention d'être inviolable, la perfection n'étant pas de ce monde. Toutefois, des attaques externes n'auraient que peu d'impact sur la sécurité des transmissions.

7. Conclusion

Résultat obtenu

Nous avons réussi à mettre en place une architecture suffisamment évoluée pour proposer une nouvelle façon conviviale de se connecter à un site internet proposant du contenu sécurisé. Cette solution a été conçue de telle sorte que l'intégration dans différents système et la maintenance soient facilitées. Les utilisateurs peuvent maintenant accéder à du contenu sécurisé sur un site en scannant simplement un QR Code avec une application dédiée sur leur Smartphone. C'est seulement sur cette dernière que l'utilisateur saisit ses identifiants personnels, rien n'est saisi sur l'ordinateur. A terme, l'intégration du Common Layer à la solution industrialisée va permettre aux sites internet de profiter de l'authentification forte LuxTrust pour s'assurer de l'identité de ses utilisateurs.

Bilan de l'expérience

Ce projet a été enrichissant car il m'a permis d'avoir une approche générale de la conception d'une solution. Je suis parti d'une idée de base et j'en ai vérifié sa pertinence en réalisant une analyse de marché. Il m'a ensuite fallu présenter ma solution à des partenaires qui allaient m'aider à la déployer dans un environnement réel. J'ai été en contact avec différents profils, aussi bien techniques que fonctionnels. Dans chaque cas, il m'a fallu adapter mon langage afin de distiller le juste niveau d'information de façon à accrocher mon interlocuteur sans le noyer dans des termes qui lui paraitraient barbares.

En ce qui concerne le développement, le projet a été riche car j'ai couvert un large domaine de technologie comme le développement mobile sous plateforme Android et sous plateforme iOS qui sont deux choses complètement différentes. Le langage de programmation change, la logique de programmation change et le matériel utilisé pour le développement n'est pas du tout le même. Je me suis également initié au développement Web avec différentes technologies comme J2EE ou Grails et les Webservice en POST ou en REST.

Mais le point le plus intéressant de ce projet a été l'encadrement de Ludovic. Ma société m'a permis d'ajouter un volet complémentaire à ma formation. L'objectif n'étant pas de

déléguer une partie de mon travail mais bien d'encadrer et de suivre une personne qui va réaliser des développements en restant dans la direction que j'aurai fixé. En ce sens, ça a été vraiment enrichissant. Ludovic achevait sa formation d'ingénieur à l'ESTINN, il lui manquait ce recul professionnel qui est nécessaire lorsqu'on réalise des développements pour un projet qui va être industrialisé. C'est une chose qu'on n'apprend pas à l'école et qui nécessite quelques années passées dans monde de la prestation de service. J'ai apprécié les nombreux échanges que nous avons eus durant notre collaboration. J'ai cherché à le faire réfléchir, à l'amener à critiquer les solutions que nous mettions en place. C'est grâce à nos débats d'idées que nous sommes arrivés à la solution aujourd'hui en production et distribuée chez nos partenaires.

Ce que j'aurai aimé faire de plus

Le projet est en phase de déploiement chez notre partenaire Mobey qui va l'intégrer à son produit Flashiz. Du fait de l'absence d'accord signé avec LuxTrust, l'implémentation du Common Layer n'a pas pu être intégrée au produit utilisé par Mobey. Lorsque LuxTrust aura contacté Cryptomatic et aura validé l'utilisation de leurs outils sur des plateformes mobiles, il restera à développer la solution en Objective-C pour son portage sous iOS. En effet le portage sur iOS est l'une des choses que je n'ai pas pu réaliser dans le temps impartis.

Les pistes de développement possibles

Comme dit précédemment, la solution permet une connexion à un site internet en bénéficiant de la sécurité inhérente à l'authentification LuxTrust. L'utilisateur doit donc saisir son identifiants LuxTrust, son mot de passe ainsi que l'OTP qu'il génère avec le token mis à sa disposition.

Une première évolution qu'il serait intéressant de mettre en place serait de reprendre l'algorithme qui génère les OTP pour les intégrer à l'application Smartphone. De cette façon, l'utilisateur n'aurait plus qu'à saisir son mot de passe et son identifiant. Il ne serait plus nécessaire de transporter le token.

Une autre poste d'évolution pourrait concerner le contexte d'utilisation de la solution. La connexion à un site internet pourrait devenir une authentification dans une banque ou une administration. On pourrait imaginer des bornes génératrices de QR Code à l'entrée

des bâtiments. Un visiteur pourrait scanner le code de la borne pour être aussitôt authentifié dans le service.

Il serait également possible d'utiliser l'authentification pour identifier le visiteur. Ainsi, ce dernier serait appelé par son nom et non plus par un simple numéro comme c'est le cas pour les tickets actuellement imprimés. Arrivé au bon guichet, il se trouverait face à une personne prête à le recevoir avec son dossier personnalisé déjà affiché à l'écran.

Il pourrait même être envisagé de remplacer les QR Codes par une nouvelle technologie qui commence à apparaitre sur les nouvelles générations de téléphone. Les nouveaux appareils sont équipés d'une puce RFID qui permet l'échange d'information sans contact. On appelle ceci le NFC (Near Field Communicator). Dans le cas d'une administration ou d'une banque, on pourrait imaginer que la borne qui génère les QR Code soit remplacée (ou complétée) par un capteur NFC. Un visiteur n'aurait qu'à démarrer l'application mobile, y saisir ses identifiants puis passer le téléphone à quelques centimètres du capteur pour valider l'authentification.

Si ce projet semble plus convivial et plus simple à utiliser car l'utilisateur n'a plus à utiliser la caméra de son appareil pour scanner un QR Code, il serait prématuré de le mettre en place maintenant. La technologie NFC n'est pas suffisant démocratisée et équipe à ce jour une minorité de téléphone. Il reste encore un peu de temps avant que les capteurs NFC rivalisent avec les appareils photos en termes de parts de marché.

Mon projet de stage a aussi ouvert une opportunité aux produits LuxTrust. L'implémentation du Common Layer sur des Smartphone permet l'authentification sur les serveurs d'authentification de LuxTrust mais une fois que le certificat est rapatrié sur le téléphone, tout un champ d'application s'ouvre à l'utilisateur.

En effet, en poussant le développement, il deviendra possible d'utiliser ce certificat de la même façon qu'avec un ordinateur. L'utilisateur pour chiffrer ou déchiffrer des messages de type mail, SMS ou MMS ou encore signer des documents rédigés ou ouverts sur son Smartphone.

"Toute la puissance de l'authentification forte sur un appareil qu'on a toujours sur soi"

L'ouverture du projet

Nous avons déjà réfléchi aux applications possibles pour notre solution. InTech souhaite contacter les administrations et proposer notre principe de connexion par mobile pour remplacer les actuelles files d'attentes par tickets papiers. Imaginons que les distributeurs de numéros soient remplacés par des QR Codes voire des balises NFC. Un usager pourrait alors utiliser son téléphone portable de type Smartphone pour s'authentifier dans l'administration. En fonction du degré de sécurité requis on pourrait intégrer ou pas l'implémentation du Common Layer. Dans tous les cas, le préposé au guichet n'appellerait plus un numéro mais une personne par son nom. Et lorsque l'usager se présenterait, il se trouverait face à une personne qui a déjà le bon dossier sous les yeux.

La mise en place d'une telle architecture est réalisable de cette façon :

- L'administration doit utiliser notre librairie "site client" pour l'appareil qui sera scanné par les usagers,
- Elle doit également mettre en place un Serveur d'authentification si elle ne souhaite pas intégrer l'authentification LuxTrust,
- Enfin, elle doit mettre à disposition de ses usagers une application mobile facilement téléchargeable via les plateformes de diffusion standards (Android Market, Apple Store, etc)

Les projets à venir qui vont rentabiliser les connaissances acquises

Fort des connaissances acquises en développement mobile, je pense me positionner en élément idéal pour les projets mobiles qui s'offrent à InTech. Pour n'en citer que deux, nous avons un partenariat avec les ACL (Automobiles Clubs Luxembourgeois) pour développer de nouveaux outils qui vont fédérer les chauffeurs de taxis de Luxembourgville. Le système pourrait être adapté pour faciliter le travail de la Croix Rouge Luxembourgeoise ainsi que d'une société d'ambulance. Le principe sera de proposer de nouveaux services utilisables par un opérateur (via une application web) et de permettre à ce dernier de communiquer des informations à ses collaborateurs (que ce soit un taxi ou une ambulance) pour lui indiquer sa prochaine destination ou lui transmettre les dernières informations sur le trafic.

Mon évolution de carrière :

Durant toute cette expérience j'ai pris beaucoup de plaisir à aller plus loin que le simple développement. Analyser un contexte économique pour définir le meilleur cadre pour un projet, étudier les moyens à disposition pour proposer la meilleure solution, avoir des échanges avec des personnes venant d'horizons différents (aussi bien technique que fonctionnel) et avoir autant de points de vue différents. Ceux sont ces étapes qui m'ont intéressé. Je souhaiterais évoluer vers un poste à responsabilité afin d'aller chercher les informations auprès d'un groupe d'utilisateur pour ensuite expliquer ma vision des choses à des profils techniques. En suivant ceci, ma carrière pourrait évoluer vers un poste de Responsable de Développement. J'aime être à l'écoute d'utilisateurs fonctionnels puis réfléchir pour apporter ma sensibilité et mon savoir-faire à des équipes de développement.

Table des illustrations

Bibliographie

[Allen S., Graupera V. et Lundrigan L., 2010]. Pro Smartphone Cross-Platform Development. Apress , 2010

[Audibert L., 2009]. *UML 2 : De l'apprentissage à la pratique.* Ellipses Marketing, 2009

[Castle T. L., 2002]. Documentation d'utilisation de la bibliothèque BouncyCastle (Bibliothèque de chiffrement). The Legion of the Bouncy Castle. http://www.bouncycastle.org/wiki/display/JA1/Home, 2002

[Catteau B. et Faugout N., 2009]. AJAX, Le guide complet. Micro Application, 2009

[Chatelier P., 2009]. *De C++ à Objective-C, 2009*

[Chauvet JM., 2002]. Services Web avec SOAP, WSDL, UDDI, ebXML. Eyrolles, 2002

[Dupuy E., 2008]. Documentation d'installation du décompilateur Java. JD-Eclipse - Java Decompiler. http://java.decompiler.free.fr/?q=jdeclipse, 2008

[Estimbre T., 2011]. Android dépasse iOS en Europe. Article du journal web Presse-Citron du 14 septembre 2011. http://www.presse-citron.net/android-depasse-ios-en-europe, 2011

[Foundation, 2012]. Documentation de l'environnement Eclipse. The Eclipse Foundation open source community website. http://help.eclipse.org/indigo/index.jsp, 2012

[Fron A., 2007]. Architectures réparties en Java. Dunod, 2007

[Gantaume B., 2011]. JUnit Mise en oeuvre pour automatiser les tests en Java. Eni, 2011

[Grosso W., 2001]. Java RMI. O'Reilly Media, 2001

[Guignard D., 2011]. *Programmation Android - De la conception au déploiement avec le SDK Google Android 2.* Eyrolles, 2011

[Lafosse J., 2011]. Développement d'applications n-tiers avec la plate-forme Java EE. Eni, 2011

[Maesano L., 2003]. *Services Web avec J2EE et .NET : Conception et implémentations.* Eyrolles , 2003

[Napier R. et Kumar M., 2012].IOS 5 - Développement d'applications avancées. Pearson Education, 2012

[Owen S., 2007]. Site de téléchargement et de documentation de l'outil ZXing (bibliothèque de capture et de traitement de l'image). http://code.google.com/p/zxing/w/list, 2007

[Pujolle G., 2000]. *Les Réseaux.* Eyrolles, 2000

[Richardson L. et Ruby S., 2007]. Restful Web Services, O'Reilly Media, 2007

[Schneier B., 2001]. Cryptographie appliquée. Vuibert, 2001

[Smartsheet, 2006]. Gestion de projet avec diagrammes de Gantt interactif en ligne. http://www.smartsheet.fr/diagramme-de-gantt-en-ligne?s=127&c=3&m=414&a=003&k=graphique%20de%20gantt&gclid=CIO9jK-MgLMCFYTMtAodq0UA7Q, 2006

[Tyley R., 2012]. Documentation d'utilisation de la bibliothèque SpongyCastle (Bibliothèque de chiffrement adaptée aux Smartphone). Roberto Tyley. https://github.com/rtyley/spongycastle, 2012

Annexe A : Les outils de chiffrements

A.1. Les certificats

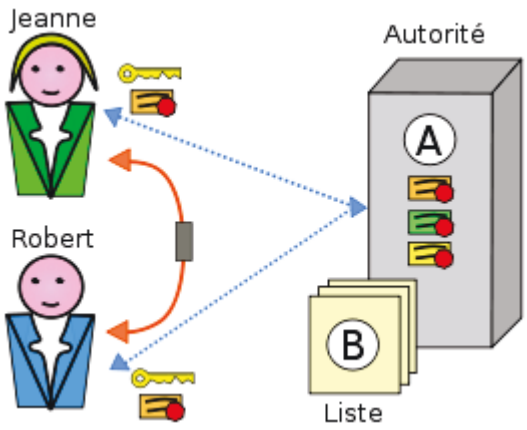

Figure 54 - Principes de l'utilisation des certificats

Un certificat [Pujolle G., 2000] est un fichier qui contient la clé publique ainsi que d'autres informations relatives à son propriétaire. Le certificat ajoute une notion de confiance à un référant, générateur du certificat racine.

L'autorité d'enregistrement et de certification (repère A) recense et contrôle l'utilisation des certificats. Il possède une liste (repère B) des certificats révoqués (figure 54).

Si Jeanne et Robert échangent leurs certificats, ils peuvent à tout moment vérifier la validité du certificat reçu auprès de l'autorité. L'autorité de certification attribue un certificat liant une clé publique à un nom distinctif (*Distinguished Name*), à une adresse électronique ou un enregistrement DNS. Ce certificat est signé à l'aide d'une paire clé publique/clé privée appartenant à l'autorité de certification (on parle de certificat racine.) Les certificats racines sont des clés publiques non signées, ou auto-signées, dans lesquels repose la confiance. Des autorités de certification commerciales détiennent des certificats racines présents dans de nombreux logiciels, par exemple les navigateurs Web. Quand le navigateur ouvre une connexion sécurisée (SSL) vers un site ayant acheté une certification auprès d'une autorité connue, il considère le site comme sûr dans la mesure où le chemin de certification est validé. Le passage en mode sécurisé est alors transparent. La durée de validité des certifications commerciales n'est pas infinie, elles expirent souvent au bout d'un ou trois ans et doivent être renouvelées. Il existe plusieurs types de certificat :

- Les certificats SSL : SSL est un protocole standard pour sécuriser des transactions en ligne. Des transferts de données via une connexion SSL sont encryptés et seulement lisibles pour ceux qui les envoient et ceux qui les réceptionnent. Le terme **SSL** se définit par **S**ecure **S**ockets **L**ayer

Un certificat SSL protège seulement un FQDN (Fully Qualified Domain Name / nom de domaine qualifié)

- Les certificats SAN : Les Certificats SSL de type Subject Alternative Name (SAN) sont également connus en tant que Certificat SSL de type Unified Communications (UCC SSL). Ces certificats autorisent des serveurs multiples ou des noms de domaine/sous-domaines utilisant le même certificat SSL sécurisé.
- Les certificats Wildcard : Il s'agit d'un seul certificat SSL pour sécuriser un nombre illimité de sous-domaines.

Ces certificats sont utilisés pour l'authentification ou pour la signature électronique.

- Les certificats objets : Techniquement un certificat objet est identique à un certificat SSL. Seul un bit change (le bit KeyUsage) pour indiquer que le certificat sert à signer du code ou du contenu de la même façon qu'un emballage plastique protège un produit.

A.2. Les signatures électroniques

La signature électronique [Schneier B., 2001] est un mécanisme permettant de garantir l'intégrité d'un document électronique et d'en authentifier l'auteur, par analogie avec la signature manuscrite d'un document papier. Un mécanisme de signature électronique doit présenter les propriétés suivantes :

- authentification : l'identité du signataire doit pouvoir être retrouvée de manière certaine.
- infalsifiable : la signature ne peut pas être falsifiée. Quelqu'un ne peut se faire passer pour un autre.
- non réutilisable: la signature n'est pas réutilisable. Elle fait partie du document signé et ne peut être déplacée sur un autre document.
- intégrité : un document signé est inaltérable. Une fois qu'il est signé, on ne peut plus le modifier.
- non répudiation : la personne qui a signé ne peut le nier.

Elle se différencie de la signature écrite par le fait qu'elle n'est pas visuelle, mais correspond à une suite de nombres. La signature électronique n'est devenue possible qu'avec la cryptographie asymétrique (voir plus loin).

Supposons que l'on dispose d'un algorithme de chiffrement asymétrique. Notons C, la fonction de chiffrement et D celle de déchiffrement. La fonction C est capable de chiffrer

une information "claire". La fonction D ne peut que déchiffrer une information préalablement chiffrée par C

C est "fournie" par la clé privée, et n'est donc connue que du propriétaire légitime.
D est "fournie" par la clé publique, et ainsi possiblement connue par tous.

Prenons maintenant un message M que nous souhaiterions signer. L'idée est d'utiliser une fonction H de hachage (par exemple MD5 ou SHA). Le résultat H(M) de cette opération permet de s'assurer de l'intégrité du document, qu'il est bien entier et sans erreur. Ce condensat est chiffré par C. C'est ce condensé chiffré C(H(M)) qui constitue la signature du message. Une personne qui reçoit le message M et sa signature C(H(M)) peut utiliser la clef publique (donc D) pour décrypter le condensat chiffré.
Le condensat H(M) peut être généré par n'importe qui tant que la même fonction de hachage est utilisée. Il ne reste alors qu'à comparer H(M) avec D(C(H(M))).

Si D annule bien C et que les condensats sont identiques, cela valide que la signature a été chiffrée par notre clef privée, l'authentification et l'intégrité du message sont donc garanties. i les condensats sont différents, cela signifie que la signature n'est pas la nôtre ou que le message a été altéré.

A.3. Cryptographie Symétrique

Cette méthode (figure 55) est la plus simple, deux utilisateurs (Anne et Bob) qui souhaitent communiquer vont chiffrer leur message avec la même clef (clef A).

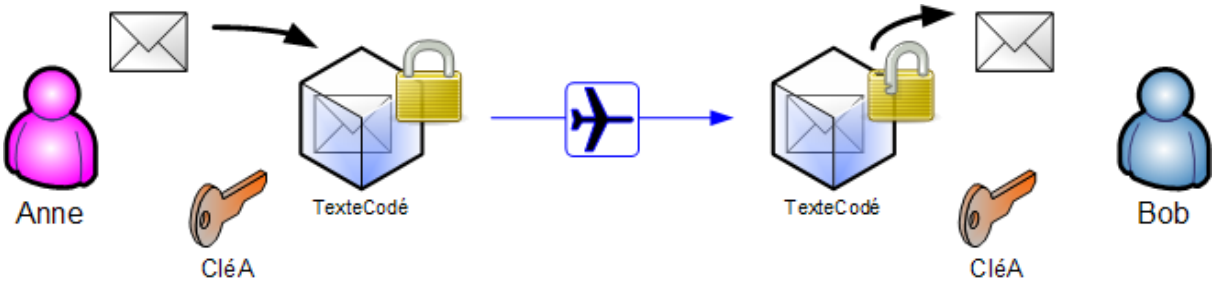

Figure 55 - Cryptographie Symétrique

Cette méthode est limitée lorsque l'utilisateur décide de communiquer avec plusieurs destinataires et de leur attribuer des clefs différentes. De plus la clef peut tout à fait être interceptée par un tiers qui pourrait l'utiliser pour déchiffrer les messages.

A.4. Cryptographie Asymétrique

Cette méthode (figure 56) fait appel aux mathématiques. On parle d'algorithmes asymétriques car les clefs utilisées pour chiffrer et déchiffrer un message ne sont pas les mêmes. Chaque utilisateur possède une paire de clef. La clef publique est diffusée via le certificat, l'autre clef, la clef privée, est conservé par l'émetteur. La propriété des algorithmes asymétriques est qu'un message codé par une clé publique n'est lisible que par le propriétaire de la clé privée correspondante. A l'inverse, un message chiffré par la clé privée sera lisible par tous ceux qui possèdent la clé publique.

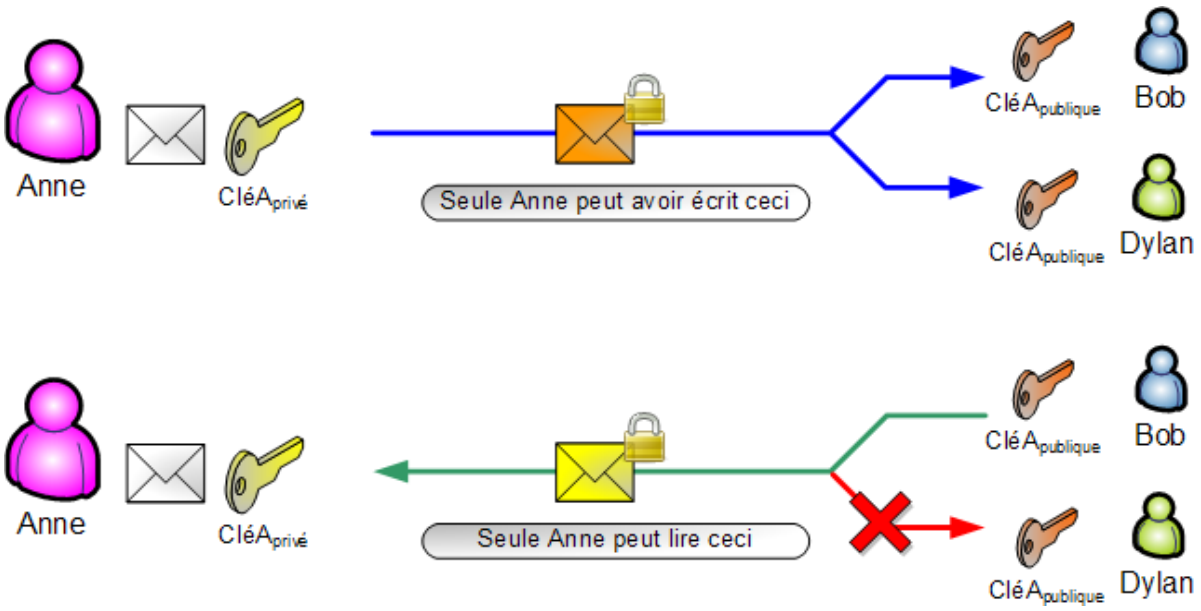

Figure 56 - Cryptographie Asymétrique

Dans le schéma ci-dessus, on peut noter que Bob sera incapable de déchiffrer le message qu'il a envoyé à Alice. Il lui faudrait la clef privée de cette dernière.

A.5. Cryptographie Mixte

Les algorithmes de cryptographie asymétrique sont plus lents. Le principe de la cryptographie mixte est donc de choisir au hasard une clef secrète et de chiffrer le

message avec. C'est ensuite cette clef qui sera chiffrée avec la clef publique du destinataire et non tout le message. La cryptographie mixte associe donc la sécurité des bi-clefs à la performance des algorithmes symétriques.

A.6. Les algorithmes de chiffrement les plus courants

A.6.1. DES (Data Encryption Standard)

L'algorithme DES [Schneier B., 2001] a été créé dans les laboratoires de la firme IBM Corp. Il est devenu le standard du NIST (National Institute of Standards and Technology) en 1976 et a été adopté par le gouvernement américain en 1977. C'est un chiffrement qui transforme des blocs de 64 bits avec une clé secrète de 56 bits au moyen de permutations et de substitutions. Le DES est considéré comme étant raisonnablement sécuritaire. Le DES est officiellement défini dans la publication FIPS 46-3 et il est public. La clé est en fait constituée de 64 bits, dont 56 bits sont générés aléatoirement et utilisés dans l'algorithme. Les huit autres bits peuvent être utilisés pour la détection d'erreurs (dans une transmission par exemple). Chacun des huit bits est utilisé comme bit de parité des sept groupes de 8 bits. Comme blowfish, le DES est un chiffrement symétrique. Il utilise les transformations de substitution et de transposition (chiffrement par produit). Il est aussi appelé Data Encryption Algorithm (DEA).

A.6.2. AES

AES [Schneier B., 2001] est le sigle d'Advanced Encryption Standard, en français « standard de chiffrement avancé ». Il s'agit d'un algorithme de chiffrement symétrique, choisi en octobre 2000 par le NIST pour être le nouveau standard de chiffrement pour les organisations du gouvernement des États-Unis. Il est issu d'un appel d'offre international lancé en janvier 1997 et ayant reçu 15 propositions. Parmi ces 15 algorithmes, 5 furent choisis pour une évaluation plus poussée en avril 1999, MARS, RC6, Rijndael, Serpent, et Twofish. Au bout de cette évaluation, ce fut finalement le candidat Rijndael, du nom de ses deux concepteurs Joan Daemen et Vincent Rijmen (tous les deux de nationalité belge) qui a été choisi. Ces deux experts en cryptographie étaient déjà les auteurs d'un autre algorithme : Square. Le terme d'AES remplace désormais celui de Rijndael mais l'algorithme n'a pas été modifié. AES n'est toutefois qu'un sous-ensemble de Rijndael

puisque il ne travaille qu'avec des blocs de 128 bits alors que Rijndael offre des tailles de blocs et de clefs qui sont des multiples de 32 (compris entre 128 et 256 bits).
Ce faisant, l'AES remplace le DES (choisi comme standard dans les années 1970) qui de nos jours devenait obsolète, car il utilisait des clefs de 56 bits seulement. L'AES a été adopté par le NIST en 2001. De plus, son utilisation est très pratique car il consomme peu de mémoire et il est plus facile à implémenter.

A.6.4. Blowfish

Blowfish a été conçu par Bruce Schneier en 1993 comme étant une alternative aux algorithmes existants, en étant rapide et gratuit. Blowfish est sensiblement plus rapide que le DES. Il utilise itérativement une fonction de chiffrement 16 fois. La grandeur des blocs est de 64 bits. Il peut prendre une longueur de clé variant entre 32 bits et 448 bits. Depuis sa conception il a été grandement analysé et est aujourd'hui considéré comme étant un algorithme de chiffrement robuste. Il n'est pas breveté et ainsi son utilisation est libre et gratuite.

A.6.5. RSA (Rivest Shamir Adleman)

RSA [Schneier B., 2001] est un algorithme de cryptographie asymétrique, très utilisé dans le commerce électronique, et plus généralement pour échanger des données confidentielles sur Internet. Cet algorithme a été décrit en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman. RSA a été breveté par le Massachusetts Institute of Technology (MIT) en 1983 aux États-Unis. Le brevet a expiré le 21 septembre 2000. Cet algorithme est fondé sur l'utilisation d'une paire de clés composée d'une clé publique pour chiffrer (respectivement vérifier) et d'une clé privée pour déchiffrer (respectivement signer) des données confidentielles.

A.6.6. PGP (Pretty Good Privacy)

PGP a été créé en 1991 par Philip Zimmermann, un informaticien américain poursuivi par le gouvernement américain pour trafic d'armes après avoir diffusé son logiciel sur Internet, (car la cryptographie est considérée là-bas comme une "arme" interdite d'exportation). PGP est principalement utilisée dans les cryptographies hybrides.

A.7. Les fonctions de hachage

Une fonction de hachage [Schneier B., 2001] est une fonction permettant d'obtenir un condensé (appelé aussi condensat ou en anglais *message digest*) d'un texte, c'est-à-dire une suite de caractères assez courte représentant le texte qu'il condense.
La fonction de hachage doit être telle qu'elle associe un et un seul condensat à un texte en clair (cela signifie que la moindre modification du document entraîne la modification de son condensat). D'autre part, il doit s'agir d'une fonction à sens unique (*one-way function*) afin qu'il soit impossible de retrouver le message original à partir du condensat. S'il existe un moyen de retrouver le message en clair, la fonction de hachage est dite "à brèche secrète".

MD4 et MD5 (Message Digest) furent développées par Ron Rivest. MD5 produit des condensats de 128 bits en travaillant les données originales par blocs de 512 bits.

SHA-2 (Secure Hash Algorithm 2) a été publié récemment. Les différences principales résident dans les tailles de condensat possibles : 256, 384 ou 512 bits. Il sera bientôt la nouvelle référence en termes de fonction de hachage. La première version (SHA-1) produisait des condensats de 160 bits.

Le Common Layer de LuxTrust utilise l'algorithme RSA pour chiffrer et déchiffrer les données. Lorsqu'une information doit être signée, c'est avec une fonction de hachage de type SHA que l'opération est réalisée.

Annexe B : Fichier Descriptif du Webservice

Le fichier descriptif du web service est accessible à l'adresse http://serveur-projet.kaminski.lu:8080/Authentication/Authentication?WSDL. Ce descriptif liste les différents paramètres qui composent le webservice ainsi que les méthode qui peuvent être appelées (en rouge dans le code ci-dessous). [Lafosse J., 2011]

```
<!--
Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-
WS RI 2.2.1-hudson-28-.
-->
<!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-
WS RI 2.2.1-hudson-28-.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd"xmlns:wsp="http://www.w3.org/ns/ws-
policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"xmlns:ws
am="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schem
as.xmlsoap.org/wsdl/soap/"xmlns:tns="http://authenticationserver.intech.lu/
" xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns="http://schemas.xmlsoap
.org/wsdl/" targetNamespace="http://authenticationserver.intech.lu/" name="
Authentication">
```

[…]

```
<types>
<xsd:schema>
<xsd:import namespace="http://authenticationserver.intech.lu/" schemaLocati
on="http://serveur-
projet.kaminski.lu:8080/Authentication/Authentication?xsd=1"/>
</xsd:schema>
</types>
<message name="getToken">
<part name="parameters" element="tns:getToken"/>
</message>
<message name="getTokenResponse">
<part name="parameters" element="tns:getTokenResponse"/>
</message>
<message name="authenticate">
<part name="parameters" element="tns:authenticate"/>
</message>
```

```
<message name="authenticateResponse">
<part name="parameters" element="tns:authenticateResponse"/>
</message>
<message name="verifyAuthentication">
<part name="parameters" element="tns:verifyAuthentication"/>
</message>
```

[…]

```
</operation>
<operation name="verifyAuthentication">
<wsp:PolicyReference URI="#AuthenticationPortBinding_verifyAuthentication_W
SAT_Policy"/>
<soap:operation soapAction=""/>
<input>
<wsp:PolicyReference URI="#AuthenticationPortBinding_verifyAuthentication_W
SAT_Policy"/>
<soap:body use="literal"/>
</input>
<output>
<wsp:PolicyReference URI="#AuthenticationPortBinding_verifyAuthentication_W
SAT_Policy"/>
<soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="Authentication">
<port name="AuthenticationPort" binding="tns:AuthenticationPortBinding">
<soap:address location="http://serveur-
projet.kaminski.lu:8080/Authentication/Authentication"/>
</port>
</service>
</definitions>
```

Annexe C : ServerAPI.h et ServerAPI.m

La classe ServerAPI.h est l'interface qui va définir les attributs et les méthodes qui pourront être appelées par l'application iPhone [Napier R. et Kumar M., 2012] [Chatelier P., 2009]. Dans le cas ci-dessous, l'interface définie une variable globale initialisée avec l'adresse du serveur qui héberge le webservice, une méthode initValues et une méthode connect qui implémentera l'appel au webservice.

```
//
//  ServerAPI.h
//
#import <Foundation/Foundation.h>
@interface ServerAPI : NSObject {
}
+ (NSString *) serverUrl;
+ (void) initValues;
+ (NSString *)connect:(NSString *)server user:(NSString *)theUser token:(NSString *)theToken;
@end
```

La classe ServerAPI.m va implementer les methods définies dans l'interface. Concernant la méthode "connect" l'appel à un webservice n'est pas disponible nativement. Il a donc fallu la réécrire en composant l'adresse (url) du webservice avec l'adresse du serveur, les paramètres à passer ainsi que leurs valeurs. Le résultat obtenu est exécuté comme une consultation de page internet. Au lieu de récupérer le code HTML d'une page à afficher dans le navigateur du téléphone, nous devons décomposer la chaîne de caractère reçues pour obtenir la réponse du webservice.

```
//
//  ServerAPI.m
//
#import "ServerAPI.h"
@implementation ServerAPI
static NSString *serverUrl;
+ (NSString *)serverUrl {
        return serverUrl;
}
```

```
+ (void) initValues {
   NSString *svrp;
   serverUrl = @"http://192.168.1.11:8080/SignIz/services/Authentication";
}
+(NSString *)connect:(NSString *)theServer user:(NSString *)theUser token:(NSString *)theToken
{
   serverUrl = [NSString stringWithFormat:@"http://%@:8080/SignIz/services/Authentication",
theServer];
NSString *soap = @"<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"xmlns:aut=\"http://authenticati
onserver.signiz.net/\"><soapenv:Header/><soapenv:Body><aut:authenticate>
<login>%@</login><token>%@</token><password>moa</password></aut:authenticate></soap
env:Body></soapenv:Envelope>";

NSString *post =  [NSString stringWithFormat:soap, theUser,theToken];
   NSData *postData = [post dataUsingEncoding:NSASCIIStringEncoding
allowLossyConversion:YES];
  NSString *postLength = [NSString stringWithFormat:@"%d", [postData length]];
    NSString *req = [NSString stringWithFormat:@"%@", serverUrl];
    NSLog(@"Request : %@",req);
    req = [req stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
   NSMutableURLRequest *request = [[[NSMutableURLRequest alloc] init] autorelease];
   [request setURL:[NSURL URLWithString:req]];
   [request setHTTPMethod:@"POST"];
   [request setValue:postLength forHTTPHeaderField:@"Content-Length"];
   [request setValue:@"text/xml" forHTTPHeaderField:@"Content-Type"];
   [request setHTTPBody:postData];

   NSURLResponse *response;
   NSError *error;

   NSData *data = [NSURLConnection sendSynchronousRequest:request
returningResponse:&response error:&error];
   NSString *result = [[[NSString alloc]initWithData:data encoding:NSUTF8StringEncoding]
autorelease];
   return result;
}
@end
```

Annexe D: Licences

D.1. Utilisation non commerciale de Spongy Castle[4]

Spongy Castle uses the same adaptation of the MIT X11 License as Bouncy Castle.

D.2. Utilisation non commercial de Bouncy Castle[5]

Please note: our license is an adaptation of the MIT X11 License and should be read as such.

LICENSE

Copyright (c) 2000 - 2011 The Legion Of The Bouncy Castle (http://www.bouncycastle.org)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

[4] http://rtyley.github.com/spongycastle/
[5] http://www.bouncycastle.org/licence.html

D.3. Utilisation non commerciale du package de W3C[6]

By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.

Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.

Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

Disclaimers

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Notes

This version: http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

[6] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". Otherwise, this version is the same as the previous version and is written so as to preserve the Free Software Foundation's assessment of GPL compatibility and OSI's certification under the Open Source Definition.

D.4. Utilisation non commerciale de l'API ZXing[7]

L'API est sous licence Apache v2.0. La philosophie de cette licence est d'autoriser la modification et la distribution du code sous toute forme (libre ou propriétaire, gratuit ou commercial) tout en obligeant le maintien du copyright lors de toute modification.

Dans le cadre de notre utilisation non commerciale, les seuls prérequis sont :

- Fournir une copie de la licence originale
- Indiquer (via un fichier texte CHANGE) les fichiers qui auraient été modifiés
- Reprendre le fichier NOTICE et y faire figurer les évolutions que nous aurions apportées à l'outil.

[7] http://code.google.com/p/zxing/wiki/LicenseQuestions

Résumé :

Les technologies mobiles et le nomadisme numérique affichent une croissance exponentielle sur ces dernières années. Le matériel devient de plus en plus puissant alors que les prix deviennent plus abordables. La conséquence de ceci est un véritable tournant dans l'utilisation que l'on peut faire d'un téléphone ou d'une tablette. Aujourd'hui, un utilisateur possède un appareil qui tient dans la poche mais qui lui fournit toute la puissance d'un ordinateur et l'accès à tous les services d'Internet.

L'évolution des services associés va toutefois de pair avec un besoin accru de sécurité. La sécurisation des communications est un des facteurs primordiaux avec la convivialité de l'interface.

Ce mémoire d'ingénieur s'attache à retracer les étapes qui ont été nécessaires afin d'adresser ces besoins. Un produit existant et commercialisé par LuxTrust S.A. a ainsi été adapté à des périphériques mobiles de type Smartphone afin de permettre des échanges sécurisés entre ces derniers et les serveurs d'authentification de LuxTrust. Une architecture composée d'un serveur d'authentification, d'un serveur web et d'une application mobile a donc été mise en place pour prouver la faisabilité d'un portage de la couche d'authentification, le Common Layer, sur un appareil de type Smartphone.

Mots clés : Smartphone, QR Code, sécurité, authentification, services web.

Abstract:

Mobile technologies and digital nomadism show exponential growth in recent years. The devices become more powerful as prices become more affordable. The consequence of this is a turning point in the use that can be done with a phone or a tablet. Today, a user has a device that fits in his pocket but it provides him with all the possibilities of a computer and an access to all the services of the Internet.

The evolution of related services is, however, associated with an increased need for security. Securing communications is a critical factor with the friendliness of a GUI (Graphical User Interface).

This engineering thesis endeavors to retrace the steps that were needed to address these needs. An existing product marketed by LuxTrust SA was adapted for mobile devices as Smartphone to allow secured communications between the latter and LuxTrust's authentication servers. Architecture consists of an authentication server, a web server and a mobile application has been implemented to proof the feasibility of porting the authentication layer, the Common Layer, on a Smartphone.

Keywords: Smartphone, QR Code, security, authentication, web services.