h e g

Haute école de gestion
Genève

Architecture et mise en place de services Web dans un environnement sécurisé garantissant la confidentialité des informations d'un centre de contrôle de qualité externe

Centre Suisse de Contrôle de Qualité
Schweizerisches Zentrum für Qualitätskontrolle
Centro Svizzero di Controllo della Qualità
Quality Control Centre Switzerland

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

João Diogo AMARAL RODRIGUES

Conseiller au travail de Bachelor :

Rolf HAURI, Chargé d'enseignement

Genève, 23 septembre 2016

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Hes·so GENÈVE
Haute Ecole Spécialisée de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor of Science en Informatique de gestion.

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante :

http://www.urkund.com/fr/student/392-urkund-faq

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 23 septembre 2016

João Diogo AMARAL RODRIGUES

Remerciements

Je tiens à remercier l'ensemble des personnes qui m'ont soutenu et encouragé de loin ou de près à la réalisation de ce travail de Bachelor. A ce titre je remercie :

Monsieur Rolf HAURI, directeur de ce mémoire qui a accepté de suivre ce projet tout en apportant de précieux conseils. Sa disponibilité et son écoute m'ont été très utiles pour conduire ce projet jusqu'à la fin.

Toute l'équipe du CSCQ (Centre Suisse de Contrôle de Qualité, Chêne-Bourg, Suisse) qui m'a accueilli chaleureusement dans leurs locaux pendant mon intervention. Je remercie en particulier :

Monsieur Dr. Xavier ALBE, responsable du pôle technologie au CSCQ. Il m'a permis de réaliser ce travail pour le CSCQ et m'a accordé toute sa confiance. Il m'a guidé sur toutes les étapes de mon projet, m'a fourni des documents qui m'ont été très utiles et m'a apporté son expérience d'informaticien tant au niveau du développement qu'au niveau de la documentation.

Madame Dagmar KESSELER, directrice du CSCQ qui m'a accueilli au sein de son entreprise et s'est montrée très disponible pour répondre à mes questions.

Monsieur David MOSER, informaticien au CSCQ. Il m'a aidé à mettre en place mon poste de travail ainsi que tous les outils nécessaires à la réalisation de ce travail.

Finalement je tiens à remercier ma famille et mes amis proches qui m'ont encouragé pendant ce travail, mais également tout au long de mon parcours scolaire.

Résumé

Ce travail a pour but de définir une architecture de services Web performante, évolutive et facile à maintenir puis de la mettre en place dans un environnement sécurisé. A cet effet, une analyse de différentes solutions d'architectures est présentée. Elle inclut les principaux précurseurs des architectures distribuées tels CORBA, DCOM et RMI ainsi que des technologies plus récentes, à savoir SOAP et REST.

Les choix retenus se basent sur l'analyse des avantages et inconvénients de ces différentes technologies. Ils conduisent par la suite au développement d'une application côté serveur qui permet de gérer le service Web et au développement d'une application côté client qui permet d'interagir avec ce service tant sur des terminaux mobiles que des postes de travail standard.

Ces applications sont développées pour le Centre Suisse de Contrôle de Qualité (Chêne-Bourg, Suisse) qui traite des données confidentielles. Une attention particulière est donc portée au développement de services satisfaisant cette contrainte. Dans ce cadre, une recherche parallèle est réalisée tout au long de ce travail analysant différentes solutions de sécurité permettant de garantir la confidentialité des informations accédées par un service Web.

Table des matières

Liste des tableaux

Liste des figures

1. Glossaire

La liste ci-dessous contient toutes les abréviations utilisées dans ce document. Cette dernière est triée dans l'ordre alphabétique des abréviations et chaque abréviation est suivie de sa signification.

Tableau 1 : Abréviations

Abréviation	Signification
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
B2B	Business To Business
B2C	Business To Consumer
BOA	Basic Object Adaptor
C#	C sharp (langage de programmation orienté objet)
C++	Langage de programmation compilé, orienté objet
COM	Component Object Model
CORBA	Commen Object Request Broker Architecture
CQE	Contrôle de Qualité Externe
CRUD	Create Read Update Delete
CSCQ	Centre Suisse de Contrôle de Qualité
CSV	Comma-separated values
DCE/RPC	Distributed Computing Environment/Remote Procedure Calls

DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DMZ	Demilitarized zone
DSI	Dynamic Skeleton Interface
FTP	File Transfer Protocol
GUI	Graphical User Interface
HATEOAS	Hypermedia As The Engine Of Application State
HDM	Man In The Middle Attack
HEG	Haute Ecole de Gestion de Genève
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
HUG	Hôpitaux Universitaires de Genève
IBM	International Business Machines Corporation
IDL	Interactive Data Language
IIOP	Internet Inter-ORB Protocol
ISO	Organisation internationale de normalisation
Java	Langage de programmation orienté objet
Java EE	Java Enterprise Edition
JAVA SE	Java Standard Edition

JDK	Java Development Kit
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
ODL	Object Definition Language
OMG	Object Management Group
ORB	Object Request Broker
PC	Personal Computer
POA	Portable Object Adapter
QUALAB	Commission suisse pour l'assurance de qualité dans le laboratoire médical
REST	Representational State Transfer
RMI	Java Remote Method Invocation
SI	Système d'information
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Stuctured Query Language
SSL	Secure Sockets Layer
SUN	Sun Microsystems
TCP/IP	Transmission Control Protocol/Internet Protocol

UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VB	Visual Basic, langage de programmation interprété
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	Extensible Markup Language
XQUERY	XML Query
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

2. .Introduction

2.1 Avant-propos

Pendant mes recherches de thème de travail de Bachelor à la HEG, le CSCQ m'a proposé un travail de recherche et pratique dans le domaine des services Web. Je trouve ce projet très intéressant et d'actualité ; en effet, la technologie des services Web est très demandée et c'est pour cette raison que son utilisation est en forte croissance depuis plusieurs années. Après divers contacts par email et téléphone, nous sommes arrivés à un accord et avons défini le titre du travail :

Architecture et mise en place de services Web dans un environnement sécurisé garantissant la confidentialité des informations d'un centre de contrôle de qualité externe

Le CSCQ, association organisatrice d'essais d'aptitude reconnue, propose des enquêtes de contrôle de qualité externe principalement pour les laboratoires médicaux. Ces enquêtes concernent de nombreux domaines de l'analyse médicale notamment ceux de la chimie clinique, de l'hématologie et de la microbiologie.

Dans ce cadre, le CSCQ fournit à ses adhérents des échantillons de contrôle qu'ils doivent analyser dans leur environnement de routine. Ils mesurent sur chacun des échantillons un ou plusieurs paramètres appelés analytes ou constituants. Les laboratoires envoient au CSCQ leurs résultats sous format papier ou sous format électronique. Après avoir collecté toutes les données, le CSCQ effectue une analyse statistique des résultats. Le CSCQ envoie alors à chaque participant un rapport personnalisé sur l'enquête. Ce rapport inclut la performance du laboratoire pour chacun des analytes et tout particulièrement la conformité selon les critères de la QUALAB pour les analytes que celle-ci juge obligatoirement soumis au CQE.

Une fois par année, le CSCQ envoie aux participants un certificat. Celui-ci contient un résumé de tous les analytes que le laboratoire a évalués, comprenant notamment le nombre de participations, le nombre de conformités obtenues (pour les analytes obligatoires) et enfin la conformité globale annuelle à la fois pour la participation et pour la qualité.

2.2 Objectif du travail

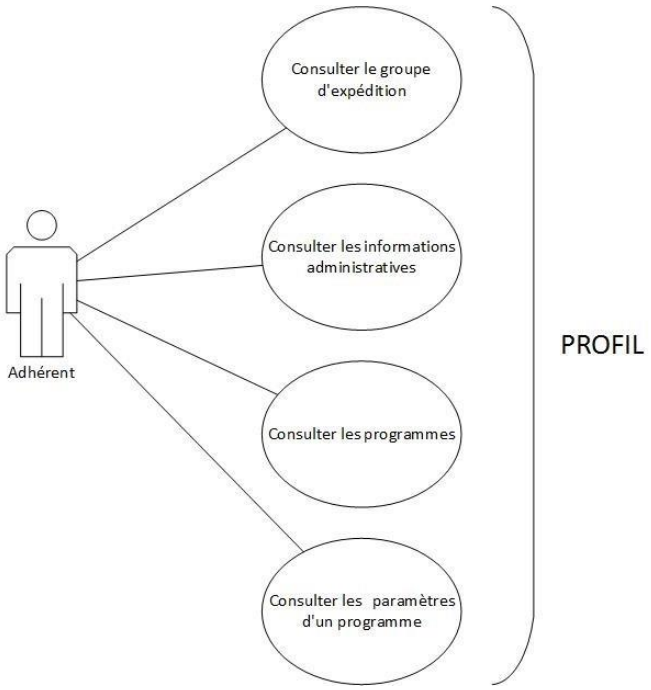
Les enquêtes proposées par le CSCQ sont subdivisées en programmes correspondant aux différents domaines des analyses médicales, par exemple la chimie clinique ou l'hématologie. Un adhérent s'inscrit à un ou plusieurs programmes et définit les

paramètres qu'il souhaite analyser. A ces informations, s'ajoutent les données administratives (nom, adresse, téléphone, email, …) et des informations relatives à l'expédition des échantillons. En effet, certaines institutions notamment les hôpitaux souhaitent regrouper l'expédition des échantillons correspondant aux enquêtes de différents services. Aussi chaque adhérent est associé à un groupe d'expédition qui comprend un à n adhérents. Les adhérents tels que les cabinets médicaux n'ont généralement qu'un seul numéro d'adhérent. Dans la suite du document, le terme « profil » inclut les données administratives, d'expédition, les programmes et les paramètres d'un adhérent dans la base de données du CSCQ.

L'objectif final de ce travail est d'offrir un service interactif supplémentaire aux adhérents du CSCQ permettant à ces derniers d'aller consulter leur profil (figure 1), c'est-à-dire :

- leur groupe d'expédition et les adhérents associés
- leurs informations administratives
- leurs programmes
- leurs paramètres ainsi que les détails associés tels que l'appareil de mesure et les réactifs

Figure 1 : Fonctionnalités du service Web



Actuellement un adhérent peut consulter son profil de deux manières différentes :

- à partir du document « Confirmation de participation » qui lui est envoyé par la poste lors de son inscription ou à chaque modification majeure de son profil
- par téléphone pendant les horaires d'ouverture du CSCQ

Le service Web offre à l'adhérent une manière plus dynamique de consulter son profil. En effet, le service est accessible en tout lieu et à tout moment, du moment qu'une connexion Internet est disponible.

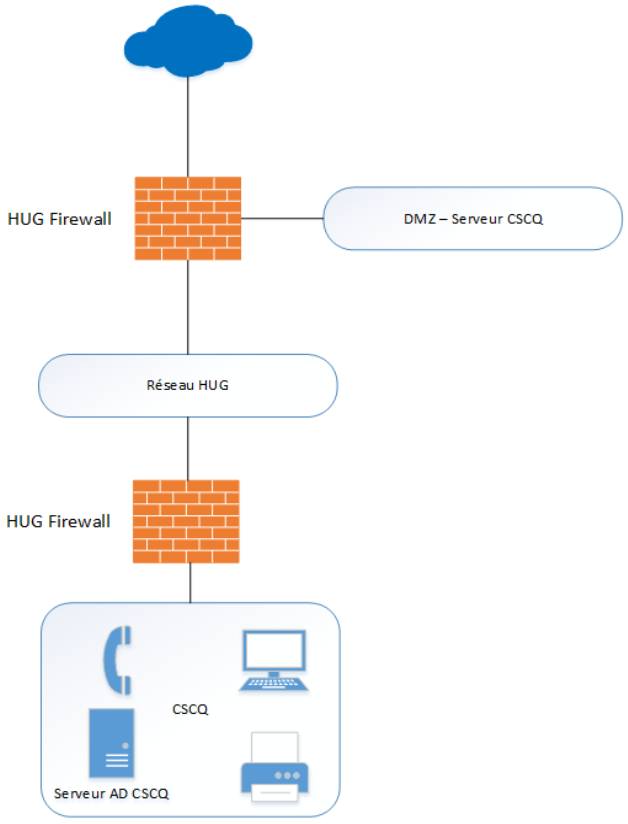
L'avantage pour l'adhérent est de pouvoir vérifier simplement la validité des informations contenues dans son profil et de pouvoir notifier au CSCQ tout changement. Le bénéfice pour l'adhérent est d'être évalué sur des informations à jour. Le bénéfice pour le CSCQ est de réduire le nombre de réclamations liées à des modifications non signalées par les adhérents.

2.3 Problématique du sujet

2.3.1 Environnement

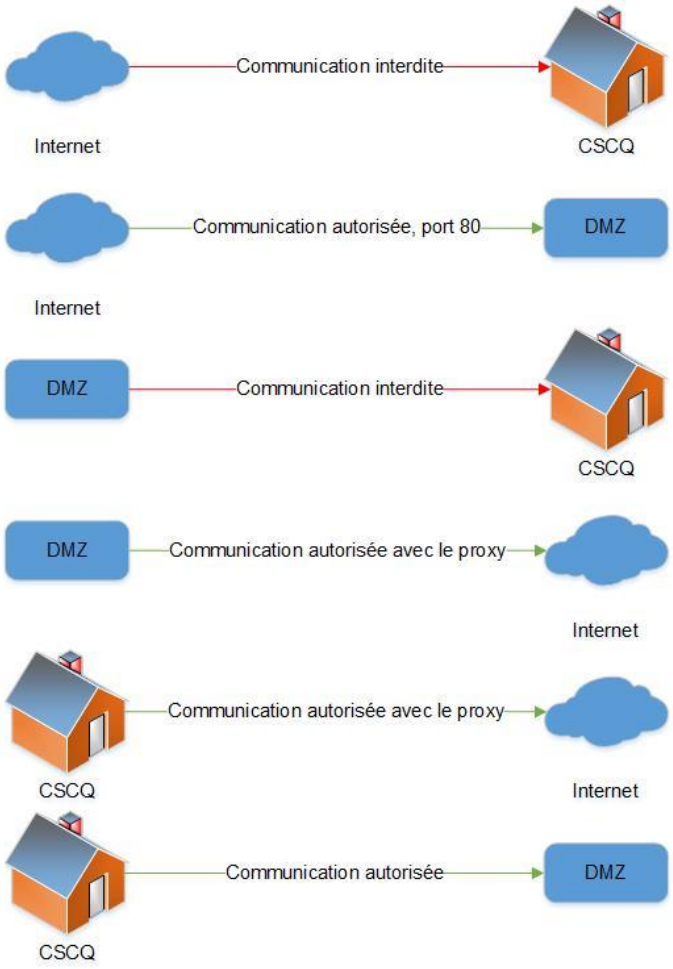
Un facteur à considérer dans ce travail est l'environnement du CSCQ. En effet, l'entreprise est logée dans les locaux des HUG et dépend du réseau informatique de ceux-ci. Le service Web à mettre en place doit avoir accès aux données du CSCQ depuis l'extérieur (Internet), or les HUG restreignent un certain nombre de communications de l'extérieur vers l'interne (CSCQ) pour des raisons de sécurité. Cependant, les communications du CSCQ vers l'extérieur fonctionnent parfaitement via un proxy. Le schéma ci-dessous est l'environnement fourni par les HUG avant la mise en place de ce travail.

Figure 2 : Environnement du CSCQ



Ce schéma montre que le CSCQ est protégé par deux pare-feux avant toute communication avec Internet. Les HUG ont mis à disposition une DMZ au niveau du premier pare-feu. Cette zone démilitarisée possède également des restrictions. En effet, le CSCQ peut communiquer avec la DMZ, mais la DMZ ne peut pas communiquer avec le CSCQ. De plus, la communication du DMZ vers l'extérieur (Internet) n'est possible qu'à l'aide du proxy des HUG et depuis Internet, la communication vers la DMZ ne peut se faire qu'uniquement via le port 80. La figure ci-dessous résume les communications autorisées et refusées avant ce travail de mémoire.

Figure 3 : Résumé des communications autorisées et refusées



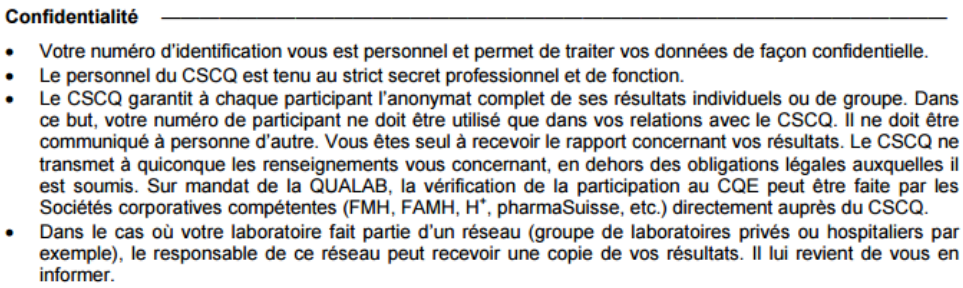
Tout au long de ce travail, une solution de communication a dû être trouvée pour qu'un utilisateur puisse accéder au service Web depuis Internet afin de consulter son profil.

2.3.2 Sécurité

La seconde problématique à traiter est la sécurité des informations, car le CSCQ garantit la confidentialité des données de ses adhérents qui est une valeur importante aux yeux

de l'entreprise et point essentiel de la norme ISO 17043, selon laquelle le CSCQ est accrédité.

Figure 4 : Extrait du CSCQ – Manuel – Section confidentialité



Confidentialité

- Votre numéro d'identification vous est personnel et permet de traiter vos données de façon confidentielle.
- Le personnel du CSCQ est tenu au strict secret professionnel et de fonction.
- Le CSCQ garantit à chaque participant l'anonymat complet de ses résultats individuels ou de groupe. Dans ce but, votre numéro de participant ne doit être utilisé que dans vos relations avec le CSCQ. Il ne doit être communiqué à personne d'autre. Vous êtes seul à recevoir le rapport concernant vos résultats. Le CSCQ ne transmet à quiconque les renseignements vous concernant, en dehors des obligations légales auxquelles il est soumis. Sur mandat de la QUALAB, la vérification de la participation au CQE peut être faite par les Sociétés corporatives compétentes (FMH, FAMH, H+, pharmaSuisse, etc.) directement auprès du CSCQ.
- Dans le cas où votre laboratoire fait partie d'un réseau (groupe de laboratoires privés ou hospitaliers par exemple), le responsable de ce réseau peut recevoir une copie de vos résultats. Il lui revient de vous en informer.

Source : CSCQ, 27 juin 2016

Dans ce cadre, le service Web doit satisfaire la contrainte de confidentialité. Un adhérent « A » doit accéder uniquement à ses informations et ne peut d'aucune manière accéder aux informations d'un adhérent « B ».

2.4 Aspects théoriques et mise en pratique

La première partie de ce document traite les différents aspects théoriques concernant les services Web. Ainsi les premiers chapitres abordent les points suivants :

- définition et fonctionnement d'un service Web
- avantages des services Web et choix des entreprises à implémenter cette technologie
- évolution des précurseurs des architectures distribuées vers les services Web
- formats de sortie utilisés
- sécurisation d'un service Web

L'aspect pratique est exposé dans la seconde partie qui décrit l'implémentation d'un service Web incluant les technologies et bibliothèques utilisées. Les solutions aux problèmes relevés à la section « Problématique du sujet » sont détaillées à ce niveau.

3. Recherche théorique

3.1 Qu'est-ce qu'un service Web ?

3.1.1 Définition

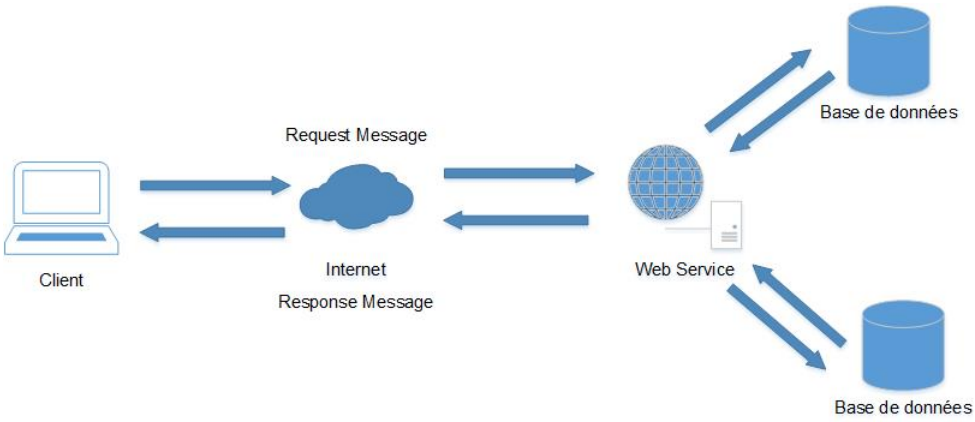
Faisant partie de la famille des technologies Web, un service Web permet à des applications et des systèmes divers de communiquer et d'échanger des données dans des environnements partagés. Cette technologie permet donc à des applications distantes de communiquer à travers un réseau (Internet) sans tenir compte des différences de langages de programmation et de plates-formes. Dans ce cadre, via un service Web, une application « A » codée en « Java » peut communiquer avec une application « B » codée en « C# ». Les requêtes et les réponses s'effectuent à l'aide de formats ouverts tels que :

- XML
- JSON
- TEXT

Un service Web repose le plus souvent sur le protocole HTTP, mais celui-ci peut également utiliser d'autres protocoles comme le FTP ou le SMTP.

Pour simplifier, un service Web fournit des interfaces qui répondent généralement à des demandes HTTP afin de fournir des données venant de différentes sources (base de données ou fichiers).

Figure 5 : Schéma service Web



Principalement utilisé comme un moyen de communication, un service Web permet à une entreprise de communiquer avec d'autres entreprises « B2B » et avec leurs clients « B2C » sans prendre connaissance des systèmes d'information se trouvant derrière le pare-feu.

Contrairement aux modèles traditionnels client/serveur (page Web), un service Web ne fournit pas à l'utilisateur une interface graphique. Un service Web permet de partager les données logiques du métier grâce à une interface de programmation à travers un réseau. Par la suite, les développeurs peuvent ajouter à ce service une GUI (page Web ou un programme) pour offrir des fonctionnalités spécifiques aux utilisateurs.

3.1.2 Fonctionnement et démarche côté client

1. Prendre connaissance des interfaces que le service Web propose.
2. Construire la requête.
3. Envoyer la requête (HTTP le plus souvent).
4. Récupérer et interpréter les données retournées (JSON, XML ou TEXT).
5. Traiter les données (calculer, afficher).

3.1.3 Fonctionnement et démarche côté serveur

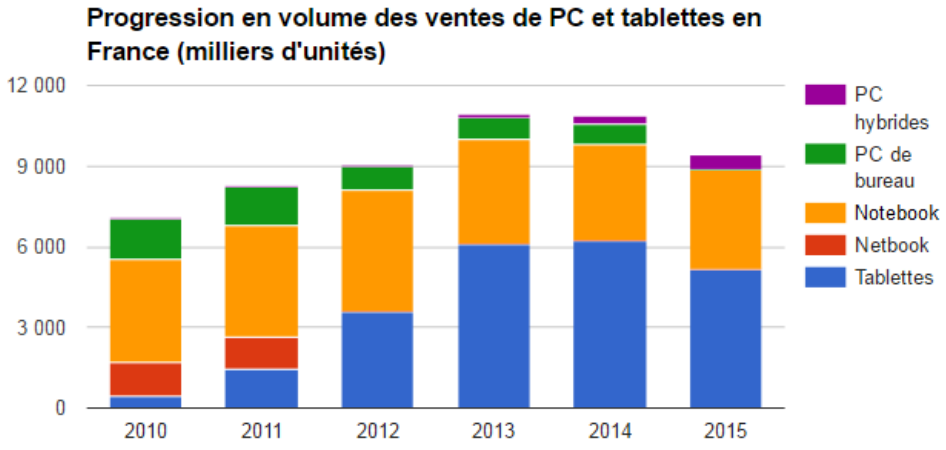
1. Définir les interfaces publiques.
2. Récupérer les requêtes du client.
3. Traduire la requête et effectuer le traitement.
4. Envoyer la réponse normalisée dans un format standard (JSON, XML ou TEXT).

3.2 Pourquoi les services Web ?

A l'heure où ces lignes sont écrites :

- il y a 3'424'971'237 utilisateurs Internet[1]
- les ventes de tablettes dépassent les ventes de PC
- les objets du quotidien deviennent connectés

Figure 6 : Ventre de PC vs Tablettes



Source : ZDNET, 28 juin 2016

[1] http://www.journaldunet.com/ebusiness/le-net/1071539-nombre-d-internautes-dans-le-monde/

Le « Big Data » n'arrête pas de s'enrichir et les demandes de services n'arrêtent pas d'augmenter. En effet, le « Big Data » ou volumes massifs de données est le terme apparu suite à l'explosion des données hébergées sur les serveurs ces dernières années. Ces données proviennent de nombreuses sources notamment des capteurs GPS des véhicules, des réseaux sociaux, des vidéos publiées en lignes ou encore des objets connectés. Il est estimé que la plus grande partie des données numériques mondiales ont été créées au cours de ces dernières années.

De plus, l'utilisateur devient de plus en plus exigeant et les entreprises doivent répondre à ses demandes pour le satisfaire. Un supermarché par exemple ne peut plus se contenter d'une grande surface de vente vers laquelle les clients se déplacent ; un supermarché doit avoir un site Internet, doit proposer un service de commande en ligne « B2C », doit échanger des informations avec d'autres entreprises « B2B » et bien plus encore. Sans oublier que les services proposés doivent fonctionner dans des environnements hétérogènes, car l'utilisateur veut consulter l'information sur tous les périphériques dont il dispose (tablette, smartphone ou PC). L'utilisateur souhaite une application simple et intuitive utilisant le moins de ressources possible.

Le service Web est une solution aux exigences ci-dessus, car il intègre une architecture adaptative et est basé sur un système de communication ouvert (interopérabilité, information accessible et interprétable par d'autres systèmes informatiques). De plus, le problème de ressources est également résolu étant donné que le client ne fait aucun traitement complexe, il se contente de lire et afficher les informations transmises. Les traitements sont décentralisés sur le serveur qui gère le service.

3.3 Les précurseurs des services Web

Les architectures distribuées sont apparues dans le monde de l'informatique depuis près de 25 ans. En effet, des technologies telles que CORBA, DCOM et RMI ont adopté ce style architectural pour distribuer de l'information entre clients, fournisseurs et partenaires. Cependant, ces technologies n'ont pas connu un énorme succès en raison de la diversité des plates-formes utilisées par les clients. De plus, elles ne sont pas adaptées à Internet, car elles ont du mal à traverser les pare-feux ce qui peut provoquer une absence de réponse. Elles nécessitent généralement un échange important de données entre le serveur et le client entraînant une certaine lenteur des appels et des réponses. Mais l'inconvénient majeur est que les précurseurs des services Web ont une architecture fortement couplée dès lors qu'une modification est faite côté serveur, une modification côté client est également nécessaire. A contrario, un service Web permet d'avoir des applications faiblement couplées ce qui favorise son utilisation et sa

maintenance. Les trois prochaines sections détaillent l'historique, les principes, les avantages et les inconvénients des technologies CORBA, DCOM et RMI.

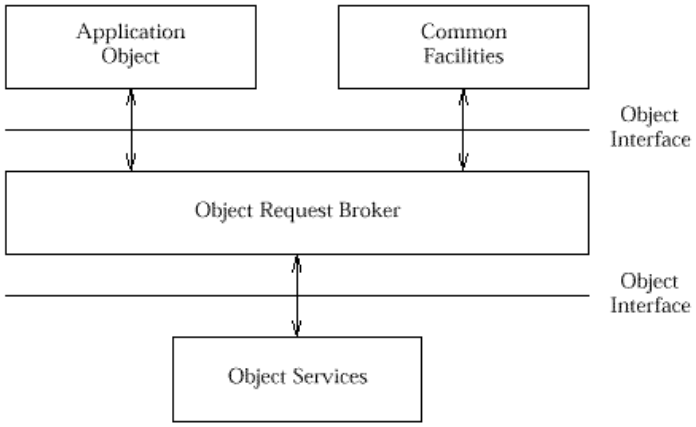
3.3.1 CORBA

3.3.1.1 Historique et principes

Tout a commencé par la fondation en 1986 de l'OMG qui regroupe des acteurs majeurs de l'informatique (Oracle, IBM, …). L'objectif de la fondation est de définir un standard et non une implémentation. Ce standard a pour but d'offrir l'interopérabilité et l'intégration de composants hétérogènes dans un système à la fois au niveau de la technologie de développement (langage de programmation) et au niveau de la plate-forme (machine). Plusieurs versions de CORBA ont vu le jour dont la première en août 1991 et la dernière en novembre 2012.

CORBA respecte la standardisation proposée par l'OMG pour les applications distribuées orientées objet, cette dernière est illustrée dans la figure suivante.

Figure 7 : OMG Objet Model



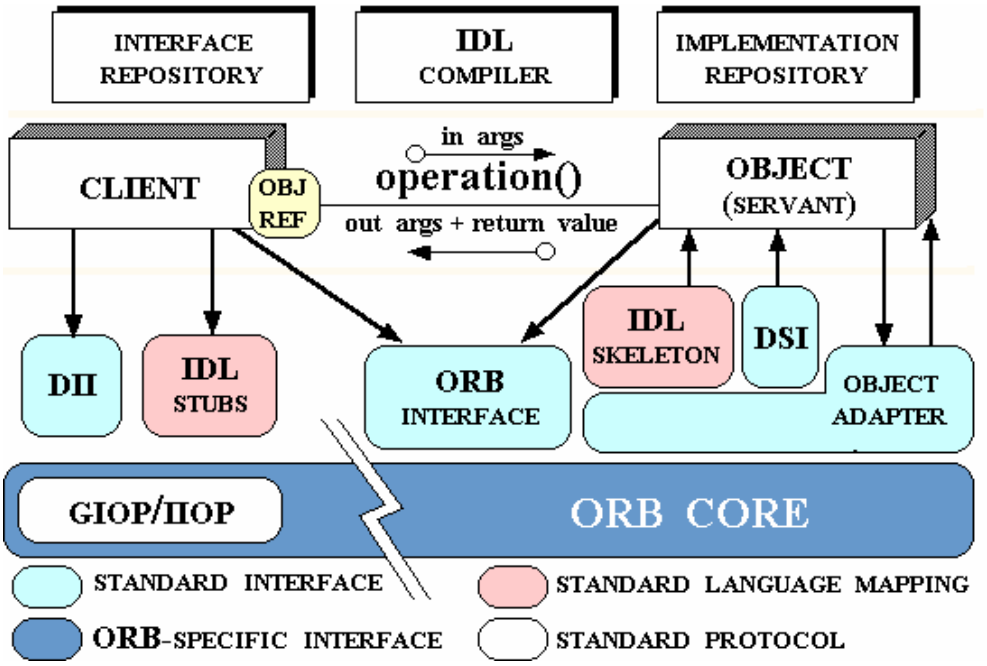
Source : THE OPEN GROUP, 16 août 2016

- « Application Object » est le composant qui permet de définir les interfaces de l'application d'utilisation finale. Cet élément ne possède pas de normalisation, car l'OMG fournit des spécifications et non une application.
- « Common Facilities » est une interface utilitaire destinée aux applications.
- « Object Request Broker » (ORB) est le cœur de la spécification CORBA. Ce composant assure l'envoi et la réception des requêtes via le protocole IIOP qui s'appuie sur la couche transport TCP/IP.
- « Object Services » fournit un certain nombre de principes pour la gestion et modélisation des objets, entre autres, le nommage et les événements.

La figure 7 montre que l'ORB est le composant le plus important de la norme de la technologie CORBA. Il assure l'interopérabilité entre les différentes applications situées

sur de différentes machines. La figure suivante représente l'architecture générale CORBA.

Figure 8 : Architecture générale CORBA



Source : LALOUM, 16 août 2016

Les composants ci-dessous assurent le mécanisme CORBA :

- « Interface Repository » (IR) est un registre contenant des informations sur les objets IDL. L'ORB utilise alors ces informations pour interpréter les valeurs retournées lors d'une requête. IDL est un langage de programmation utilisé pour définir l'interface entre l'application et un objet, pour ainsi indiquer aux clients les opérations disponibles et comment elles doivent être invoquées.
- « Implementation Repository » permet de localiser et activer les implémentations des objets. Il peut également contenir des informations supplémentaires (débogage, sécurité).
- « IDL Stubs » permet d'envoyer les requêtes.
- « IDL Skeleton » permet de recevoir les requêtes.
- « DII » permet de construire des requêtes dynamiques sans utiliser le « IDL Stubs ».
- « DSI » permet d'intercepter les requêtes dynamiques envoyées par le « DII » sans passer par le « IDL Skeleton ».
- « Objet Adapter » permet de créer, activer, implanter et maintenir les objets CORBA entre eux. Pour ce faire CORBA introduit la norme BOA dans la version 2.0, mais pour résoudre des problèmes d'interopérabilité CORBA propose ensuite la norme POA.
- « ORB Interface » permet de gérer le comportement du bus.
- « ORB CORE » est le bus (conduit) où les requêtes transitent. Ce dernier intègre le protocole IIOP.

3.3.1.2 Avantages

- L'interopérabilité
- La transparence

3.3.1.3 Inconvénients

- La complexité
- Une formation est nécessaire pour les développeurs

3.3.2 DCOM

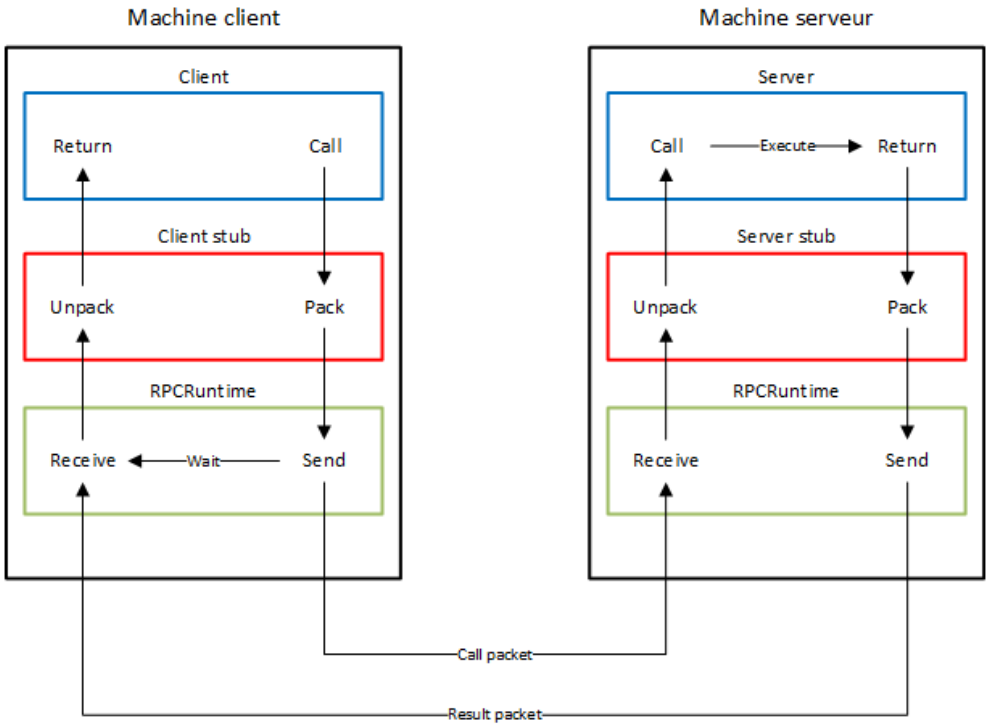
3.3.2.1 Historique et principes

DCOM est une extension de la spécification COM. Ces deux technologies ont été développées par Microsoft. COM existe depuis 1994 tandis que DCOM a vu son apparition en 1996.

COM est utilisée dans les communications internes pour lier des composants logiciels entre eux. En d'autres termes, COM permet à un composant d'être réutilisé dans un autre environnement que celui sur lequel il a été développé. Par exemple, cette technologie permet d'utiliser une feuille de calcul Excel à l'intérieur d'un document Word sans qu'Excel ne soit en fonction.

DCOM (« D » pour Distributed) est l'évolution de COM qui permet d'utiliser les spécifications de COM à travers un réseau. DCOM est le concurrent direct de CORBA, les deux technologies se ressemblent beaucoup à la différence que DCOM est adapté aux produits Microsoft. DCOM se base sur la technologie DCE/RPC qui permet de créer des applications distribuées en faisant appel à des procédures distantes. La figure ci-dessous détaille le fonctionnement de la technologie DCOM.

Figure 9 : Architecture DCOM



Source : DURAND-POUDRET, figure adaptée, 15 août 2016

1. Le client appelle la procédure.
2. L'appel de procédure est sérialisé dans un message avec les spécifications de la procédure distante et de ses paramètres.
3. Une demande est faite au « RPCRuntime » (client) de transmettre le message au « RPCRuntime » (serveur).
4. Le « RPCRuntime » (serveur) reçoit le message.
5. Le message est décodé.
6. Le serveur exécute la procédure.
7. Le résultat de la procédure est sérialisé dans un message.
8. Le « RPCRuntime » (serveur) transmet le message au « RPCRuntime » (client).
9. Le « RPCRuntime » (client) reçoit le message.
10. Le message est décodé et le résultat de la procédure est transmis au client.

3.3.2.2 Avantages

- Forte implication de Microsoft
- Rapidité
- Séparation de l'interface de l'implémentation

3.3.2.3 Inconvénients

- Dépendance de la plate-forme
- Adapté aux produits Microsoft

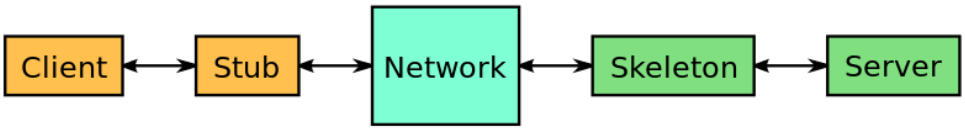
3.3.3 RMI

3.3.3.1 Historique et principes

RMI est une interface de programmation du langage Java qui permet à deux machines virtuelles de communiquer entre elles. Cette technologie a été développée par SUN et est disponible depuis la version 1.1 du JDK soit depuis 1997. RMI a pour but d'appeler, exécuter et renvoyer le résultat d'une méthode dans une machine virtuelle différente (serveur) de celle qui appelle la méthode (client).

La figure suivante décrit l'interaction entre deux machines utilisant RMI.

Figure 10 : Architecture RMI



Source : WIKIPEDIA, 15 août 2016

La technologie RMI se caractérise par deux entités le « Stub » et le « Skeleton », ces deux éléments assurent la communication entre les deux JVM.

L'objet « Stub » est implémenté côté client, ce dernier agit comme passerelle qui permet de traiter les requêtes sortantes vers le côté serveur. Les tâches suivantes décrivent le fonctionnement du « Stub » lorsqu'une méthode est appelée :

1. Initialiser une connexion vers la JVM serveur.
2. Sérialiser et envoyer les paramètres vers la JVM serveur.
3. Attendre le résultat de la JVM serveur.
4. Lire et désérialiser la valeur retournée par la JVM serveur (le résultat peut être une exception en cas d'erreur).
5. Envoyer le résultat au client.

L'objet « Skeleton » est implémenté côté serveur, il permet de gérer les requêtes entrantes. Le fonctionnement de cet objet est décrit ci-dessous :

1. Lire les paramètres que l'objet « Stub » a envoyé.
2. Invoquer la méthode côté serveur.
3. Ecrire et sérialiser le résultat de la méthode.
4. Envoyer le résultat au client.

3.3.3.2 Avantages

- Le client a l'impression de travailler en local
- Interopérabilité parfaite entre applications Java
- Préserve la sécurité fournie par l'environnement Java

3.3.3.3 Inconvénients

- Utilisation exclusive de Java
- Lenteur due la sérialisation et désérialisation
- Architecture couplée

3.3.4 Comparaison

Le tableau ci-dessus résume les principales caractéristiques des technologies CORBA, DCOM et RMI.

Tableau 2 : Comparaison CORBA, DCOM et RMI

	CORBA	**DCOM**	**RMI**
Défini par	OMG	Microsoft	SUN
Plate-forme	Multi	Win32	Multi
Langage de développement	Multi	C++, VB	Java
Langage de définition	IDL	ODL	Java

3.4 Les architectures actuelles

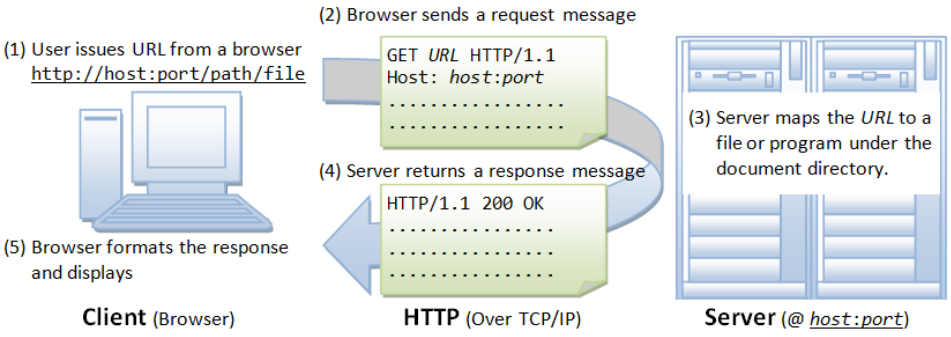
3.4.1.1 Notions de base des protocoles de communication sur le Web

HTTP est un protocole d'échange d'information sur le Web basé sur le TCP/IP. La cible d'une requête HTTP est une ressource identifiée par une URL. Une URL est composée de quatre éléments qui permettent d'identifier les ressources de manière unique sur Internet :

- le protocole (HTTP, FTP, …)
- hôte (host) (hesge.ch, cscq.ch)
- port (80, 8080)
- chemin (Path) (Chemin qui mène à la ressource sur le serveur)

La figure suivante détaille les étapes du protocole HTTP pour répondre à une requête.

Figure 11 : Fonctionnement du protocole HTTP



Source : CHUAN, 30 juin 2016

1. L'utilisateur saisit dans un navigateur le lien qu'il souhaite consulter.
2. Le navigateur envoie la demande (URL) au serveur.
3. Le serveur recherche la ressource demandée par l'URL.
4. Le serveur retourne le résultat de la demande au navigateur.
5. Le navigateur formate et affiche le résultat de la demande.

Pour la conception d'un service Web, il est important de noter comment HTTP catégorise les codes retour d'une requête. Cela permet de connaître l'état d'une requête et de trouver l'erreur rapidement en cas de problème.

- 100 à 101. Cette classe représente les codes d'informations
- 200 à 206. Cette classe représente le code de succès
- 300 à 305. Cette classe représente les codes de redirection
- 400 à 417. Cette classe représente les erreurs du client

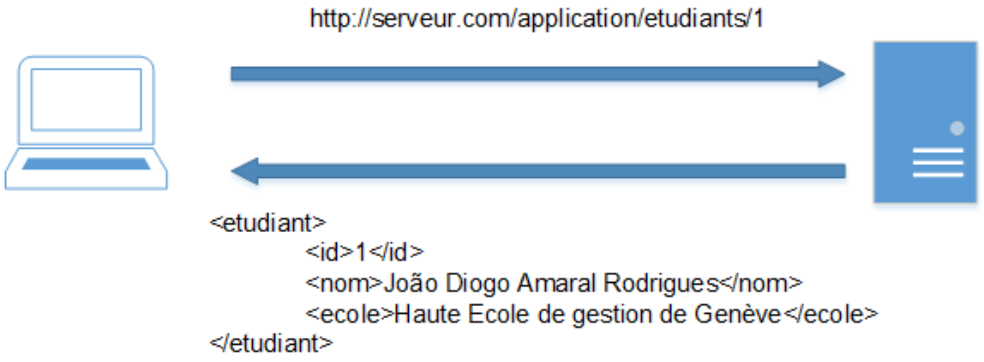
3.4.2 **500 à 505. Cette classe représente les erreurs du serveur**.

3.4.3 REST

3.4.3.1 Notions théoriques

REST n'est pas un protocole, c'est un style d'architecture utilisé pour la conception de services Web. Il a été introduit en 2000 par Roy Fiedling dans sa thèse de doctorat. Cette méthodologie est une alternative au protocole SOAP. REST fournit un ensemble de contraintes architecturales qui permettent d'accéder et de manipuler les données. L'adjectif anglais « RESTful » qualifie les systèmes qui respectent les contraintes architecturales de REST. Utilisé par les colosses de l'informatique (Google, Amazon, …), REST se base uniquement sur le protocole HTTP pour la communication client-serveur.

Figure 12 : RESTful architecture



Pour gérer le CRUD, REST s'appuie sur les méthodes HTTP suivantes :

Tableau 3 : Méthodes HTTP

Méthode	Rôle	Safe	Idempotent
GET	Permet de demander une ressource spécifique. Définit un accès en lecture. Cette demande doit récupérer les données et ne doit avoir aucun un autre effet sur la ressource.	✓	✓
POST	Permet de créer une nouvelle ressource.	✗	✗
PUT	Permet de mettre à jour l'entièreté de la ressource.	✗	✓
DELETE	Permet de supprimer une ressource.	✗	✓
PATCH	Permet de mettre à jour une partie de la ressource.	✗	✓

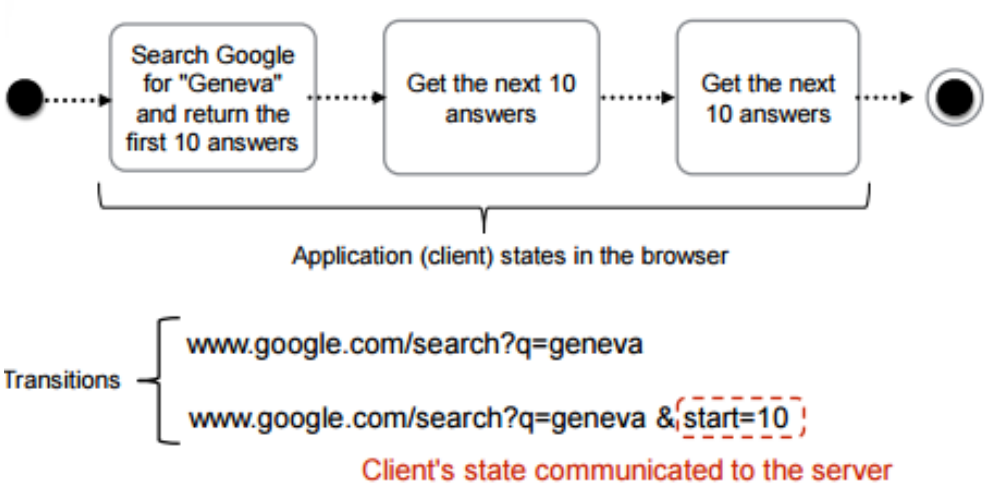
Les développeurs peuvent faire une lecture avec POST, faire une suppression avec GET ou bien faire une mise à jour avec DELETE. Mais l'application ne respecte plus les exigences REST et n'est plus « RESTful ». En effet, il est très coutumier de voir des développeurs utiliser la méthode GET pour lire, insérer et supprimer un élément alors que cette méthode est censée être utilisée uniquement pour la lecture d'information.

Une méthode est dite « safe » quand elle ne change pas l'état d'une ressource (GET). Une méthode est « idempotente » quand les traitements appliqués à plusieurs reprises produisent le même résultat. En d'autres termes, faire de multiples demandes identiques a le même effet que de faire une demande unique (GET, PUT, DELETE et PATCH).

Il est important de noter que la réponse à une requête n'est pas une ressource c'est la représentation de la ressource que le client reçoit. En effet, une ressource peut avoir plusieurs représentations (XML, JSON, HTML et TEXT). Quand le client demande une ressource, il doit définir le format de réponse (représentation) qu'il souhaite.

REST propose des services sans états (statelessness). Le serveur ne connaît aucun élément sur le client, il ne gère pas les sessions. Chaque requête est traitée de manière indépendante ce qui veut dire que la requête envoyée au serveur doit contenir toutes les informations relatives à son état. L'exemple ci-dessous illustre parfaitement le concept « statelessness » :

Figure 13 : Exemple statelessness



Source : DUGERDIL, 30 juin 2016

Dans cet exemple, la première activité permet de rechercher les dix premiers résultats avec le mot clé « Genève ». Le service ne conservant pas l'état (recherche des dix premiers résultats), la recherche des dix résultats suivants nécessite d'ajouter un paramètre (ici start=10).

Cependant, un des principes intéressants de REST est qu'il fournit en plus du résultat de la demande, des liens hypermédia permettant d'accéder à d'autres fonctions ou d'autres ressources disponibles sur le serveur (HATEOAS). Par exemple, si un appel vers le serveur demande la liste des cinq premiers étudiants alors ce dernier retourne non seulement la liste des cinq premiers étudiants, mais envoie également un lien permettant d'aller consulter les cinq prochains étudiants ou d'autres fonctions notamment la suppression ou la modification.

En résumé, REST propose des contraintes architecturales qui permettent à un client et un serveur d'échanger des données par le biais du protocole HTTP. REST n'est pas un

protocole ni une technologie à part entière, il s'agit d'un ensemble de principes et bonnes pratiques.

Dans ce cadre, REST se caractérise par les cinq principes architecturaux suivants :

- Adressabilité

  Chaque ressource est accédée par un identifiant unique décrit par une URI. Si une ressource a plusieurs représentations (JSON, XML, HTML, TEXT) alors chaque représentation a sa propre URI. C'est pour cette raison qu'une application doit construire ses URI et donc ses URL de manière précise pour satisfaire ce premier principe REST. La manière dont les URL doivent être construites est expliquée avec différents exemples dans la section « principes de base de REST».

- Interface uniform

  L'interface est représentée par les principaux verbes HTTP, ce qui évite d'inclure l'opération dans l'URI de la ressource (www.serveur.com/application/etudiants/1/delete). Dans ce cadre, une ressource est généralement représentée par les quatre opérations CRUD suivantes :

  - créer une ressource (POST)
  - lire une ressource (GET)
  - mettre à jour une ressource (PUT)
  - supprimer une ressource (DELETE)

  Il existe d'autres méthodes HTTP, mais les méthodes listées ci-dessus sont les plus utilisées.

- Orienté ressource

  Une ressource peut avoir plusieurs représentations (JSON, XML, HTML, TEXT).

- Sans état

  Chaque requête est indépendante. Une nouvelle requête doit contenir tout ce qui est nécessaire à son traitement et ne doit faire référence à aucune autre.

- Connectivité

  Permet d'avoir des liens comme relation entre les ressources.

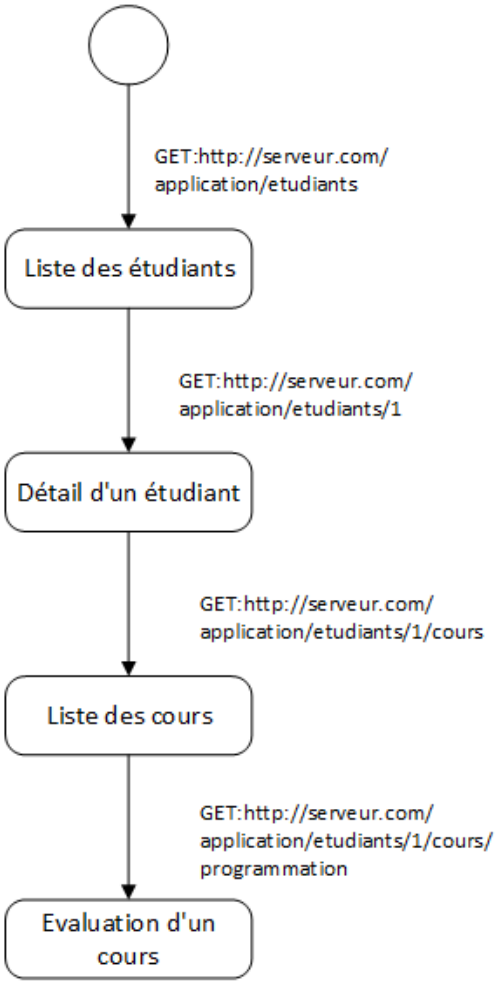
3.4.3.2 Principes de base de REST

Une bonne construction des URI est primordiale lors de la conception d'un service Web REST. Les liens (URI) dérivent de la hiérarchie des ressources et sont pour cette raison construits du plus général au plus particulier.

Prenons l'exemple d'une ressource composée d'une collection d'éléments (nommé « element ») dont chaque élément est lui-même associé avec une autre collection. Cette dernière collection contient des éléments nommés « sous-element ». Les principes REST exigent que la première collection soit appelée « elements » et que chaque membre de la collection soit nommé « element ». La syntaxe du lien permettant l'accès à l'élément de la première collection dont l'identifiant est « r » est « elements/r ». De même, la syntaxe pour l'accès à l'élément de la seconde collection dont l'identifiant est « sr » pour l'élément « r » est « elements/r/sous-elements/sr ».

Le diagramme d'activité ci-dessous explique la manière dont les URI doivent être construites à partir d'une collection d'étudiants (première collection) inscrits à une collection de cours (deuxième collection).

Figure 14 : Diagramme d'activité détaillant la construction des URI



L'URI « GET :http://serveur.com/application/etudiants » permet de récupérer la collection des étudiants. Une erreur de conception (qui n'empêche pas le service Web de fonctionner) serait d'utiliser le nom de l'objet au singulier pour récupérer une collection

« GET :http://monserveur.com/application/etudiant ». En ajoutant un paramètre à l'URI (r = le numéro de l'étudiant), l'utilisateur du service peut récupérer un étudiant spécifique « GET :http://serveur.com/application/etudiants/1 ». Le reste de la construction doit garder la même structure hiérarchique (du plus général au plus particulier). Dans ce cadre, la collection des cours de l'étudiant numéro 1 est récupérée de la manière suivante « GET :http://serveur.com/application/etudiants/1/cours ». La note d'un cours (ici programmation) est retournée quant à elle avec l'URI « GET :http://serveur.com/application/etudiants/1/cours/programmation ».

Le tableau suivant montre l'utilisation des méthodes HTTP (GET, POST, PUT, DELETE) pour accéder à des collections ou des éléments.

Tableau 4 : Résumé des méthodes HTTP dans une architecture RESTFul

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://example.com/resources`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
Element URI, such as `http://example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection		**Delete** the addressed member of the collection.

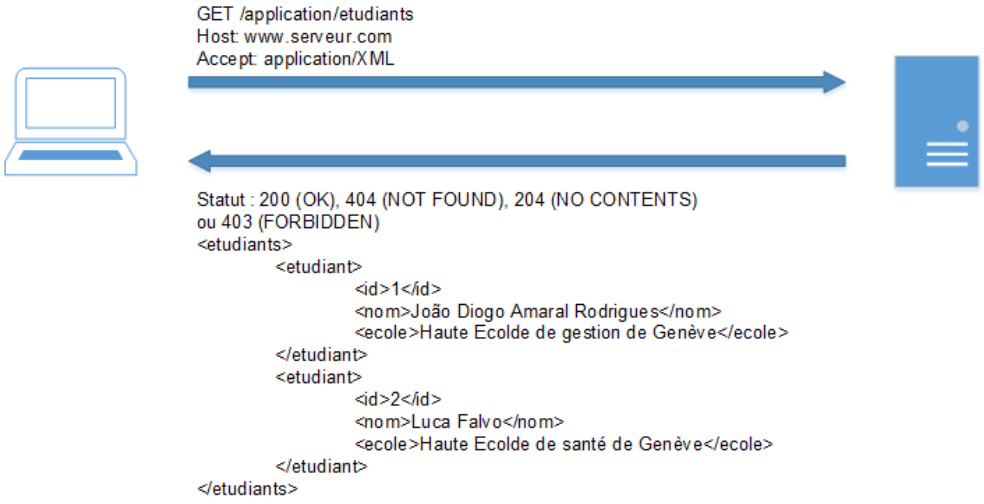
DUGERDIL, 30 juin 2016

Les principales fonctions du tableau ci-dessus sont illustrées par un exemple basé sur la collection d'étudiants utilisée précédemment.

3.4.3.2.1 Lire une collection

Pour lire une collection, l'utilisateur doit utiliser la méthode « GET » avec l'URI « GET:http://serveur.com/application/etudiants ».

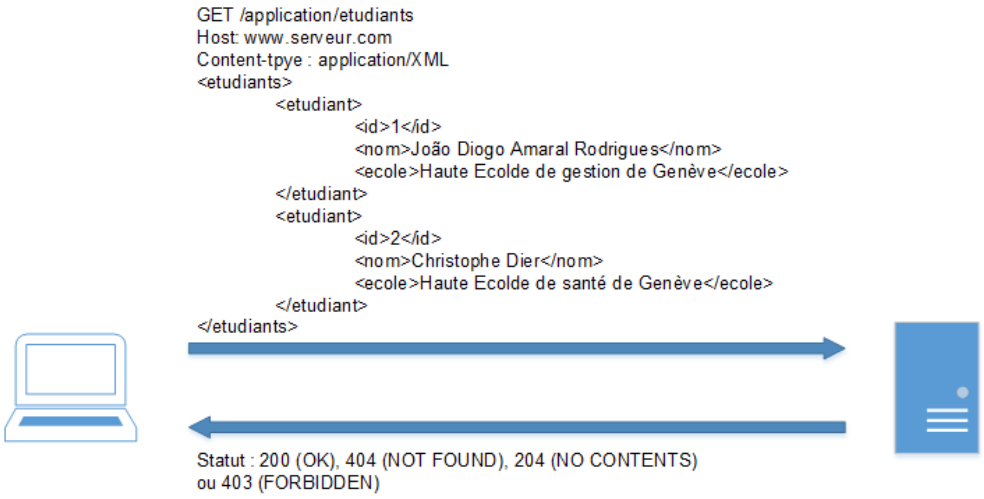
Figure 15 : Lire une collection en REST



3.4.3.2.2 Modifier une collection

Pour modifier la collection existante, l'utilisateur utilise la méthode « PUT » avec l'URI « PUT:http://serveur.com/application/etudiants » et fournit la nouvelle collection dans une représentation spécifique (XML, JSON …).

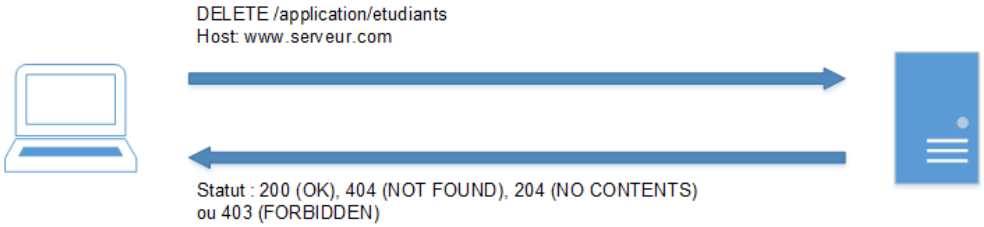
Figure 16 : Modifier une collection en REST



3.4.3.2.3 Supprimer une collection

Pour supprimer une collection complète, l'utilisateur utilise la méthode « DELETE » avec l'URI « DELETE:http://serveur.com/application/etudiants ».

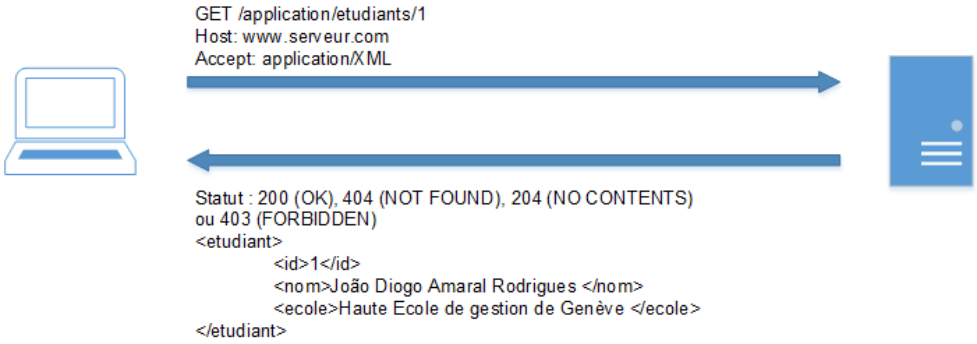
Figure 17 : Supprimer une collection en REST



3.4.3.2.4 Lire un élément d'une collection

Un utilisateur peut lire un élément de la collection avec l'URI « GET:http://serveur.com/application/etudiants/1 ».

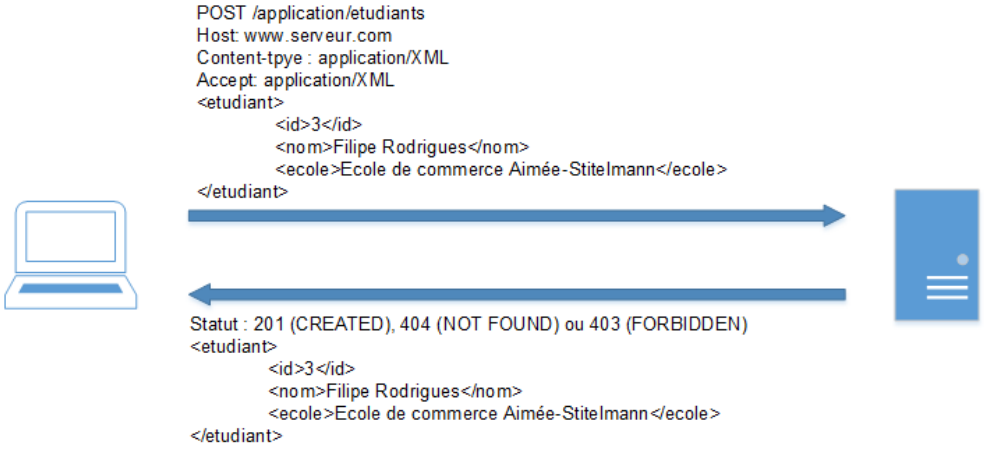
Figure 18 : Lire un élément d'une collection en REST



3.4.3.2.5 Ajouter un élément à une collection

Un utilisateur peut ajouter un nouvel étudiant à la collection avec la méthode « POST » et l'URI « POST : http://serveur.com/application/etudiants ». Pour ce faire, l'utilisateur doit fournir une représentation complète de la ressource dans le format qu'il souhaite.

Figure 19 : Ajouter un élément à une collection en REST

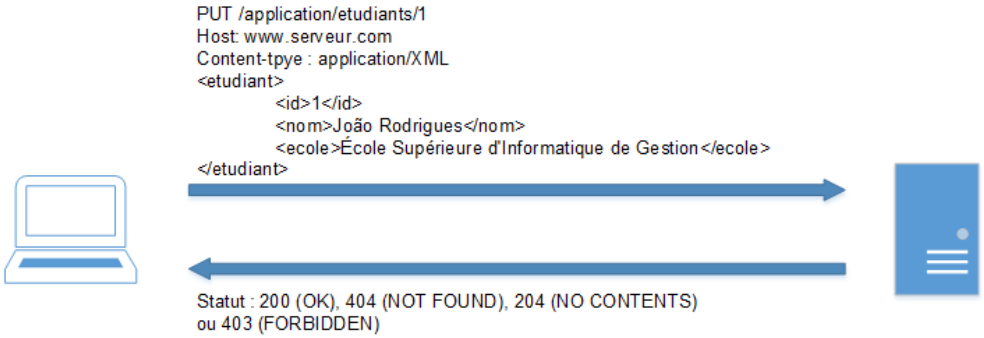


En règle générale, après une insertion, la norme veut que le service Web retourne une représentation (dans la figure ci-dessus en XML) de la ressource.

3.4.3.2.6 Modifier un élément d'une collection

Un utilisateur peut modifier une ressource de la collection avec l'URI « PUT:http://serveur.com/application/etudiants/1 ». A cet effet, l'utilisateur utilise la méthode « PUT » et fournit une représentation complète de la ressource.

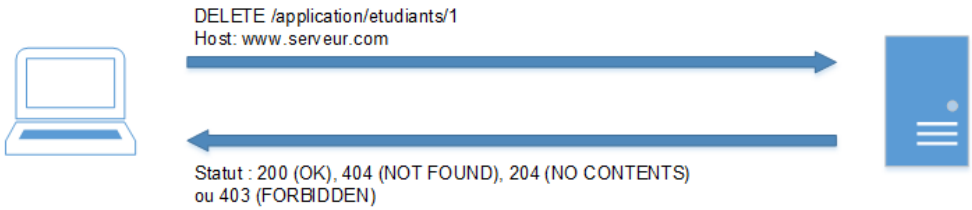
Figure 20 : Modifier un élément d'une collection en REST



3.4.3.2.7 Supprimer un élément d'une collection

La suppression d'un élément spécifique est réalisée par la méthode « DELETE » avec l'URI « DELETE:http://serveur.com/application/etudiants/1 ».

Figure 21 : Supprimer une ressource d'une collection en REST



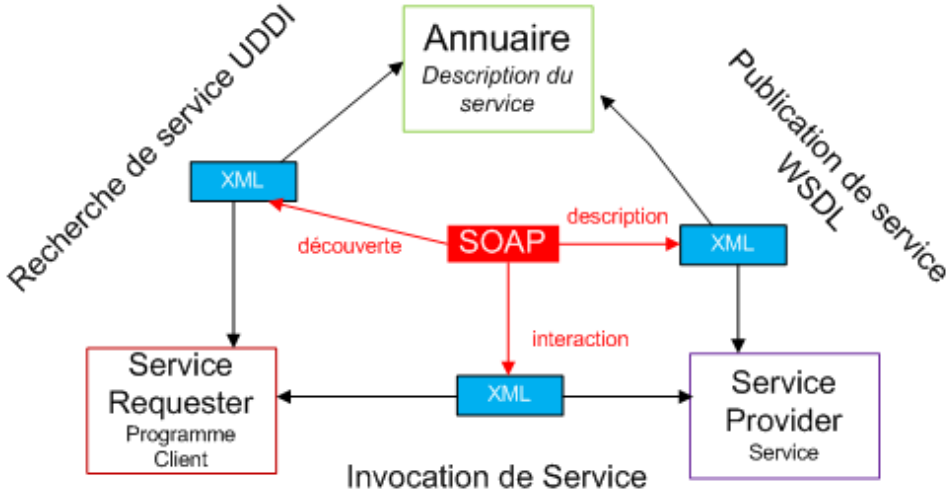
3.4.4 SOAP

3.4.4.1 Notions théoriques

SOAP est un protocole qui permet d'échanger des informations dans un système distribué basé sur XML. Il permet à une machine dite « A » d'invoquer des méthodes situées sur un autre machine dite « B » et de recevoir les résultats correspondants. SOAP n'est lié à aucun protocole de communication, mais il est généralement utilisé avec le protocole HTTP pour éviter les refus de connexion liés à la présence de proxy ou de pare-feu. La communication peut donc être basée sur d'autres protocoles tels que SMTP ou FTP. SOAP a été initialisé par Microsoft et IBM et est actuellement standardisé et recommandé par W3C.

Le fonctionnement de SOAP se base sur les trois acteurs suivants présentés sur la figure suivante.

Figure 22 : Schéma des acteurs du protocole SOAP



Source : REFES, 12 août 2016

- Le fournisseur de service (Service Provider) qui fait fonctionner le service Web et le rend disponible aux clients.
- Le client (Service Requester), consommateur du service Web. Il fait les demandes en XML à l'aide du protocole SOAP.
- L'annuaire ou registre de services. La localisation de ce registre est connue des programmeurs qui peuvent ainsi demander des services ou en publier de nouveaux.

Le flux entre ces acteurs est décrit par trois fonctions :

- publier un nouveau service. Le fournisseur de service publie dans l'annuaire la description du nouveau service.
- rechercher un service. Le client s'adresse à l'annuaire pour rechercher un service particulier. En retour, l'annuaire renvoie l'URL permettant au client d'invoquer le service.
- invoquer un service. Une fois que le client récupère la description et l'URL du service, ce dernier les utilise auprès du fournisseur de services pour invoquer les requêtes et récupérer les informations.

Cet environnement de services Web utilise plusieurs autres technologies pour former une structure à quatre couches.

Tableau 5 : Les quatre couches d'un service Web SOAP

Couche	Technologie
Découverte et publication de services	UDDI
Description de services	WSDL

Communication	SOAP
Transport	HTTP, FTP, SMTP, …

3.4.4.1.1 Découverte et publication de services

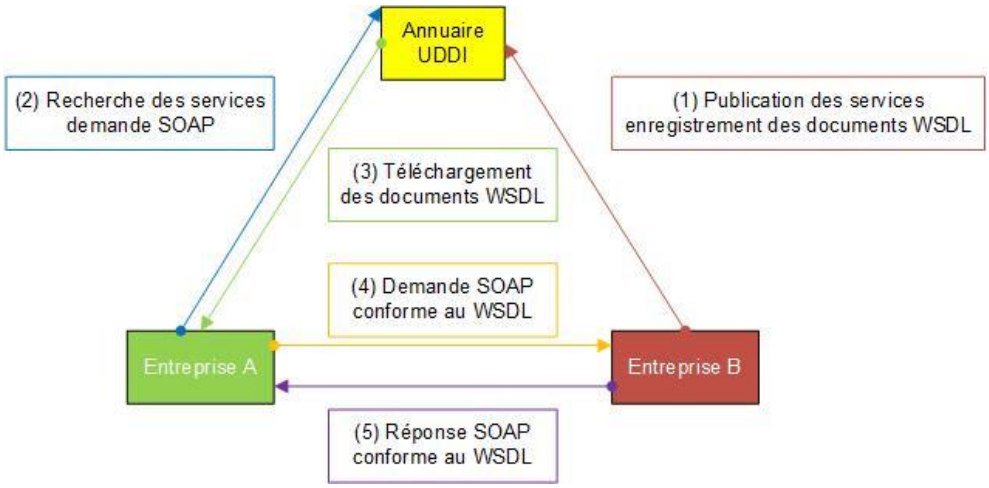
La couche découverte et publication de services permet de centraliser les services dans un registre commun permettant aux utilisateurs de rechercher rapidement les services et fonctionnalités que le service Web propose. Cette couche est assurée par un annuaire nommé UDDI, ce dernier est un standard destiné aux services Web qui permet de structurer les services proposés par les fournisseurs (entreprises). En d'autres termes, cet annuaire permet de publier et de rechercher des informations à propos des services Web. UDDI est géré par le groupe OASIS.

L'annuaire classe ses informations de consultations en trois catégories :

- les pages blanches (White Pages) qui contiennent toutes les informations liées au fournisseur (entreprise). Une page blanche contient les coordonnées, le contact ou encore la description d'une entreprise.
- les pages jaunes (Yellow Pages) contiennent les services Web que l'entreprise propose avec le standard WSDL.
- les pages vertes (Green Pages) contiennent les informations techniques et précises pour un service Web particulier.

La figure suivante explique le fonctionnement standard de l'annuaire UDDI.

Figure 23 : Schéma de l'annuaire UDDI



Source : REFES, figure adaptée, 12 août 2016

L'entreprise « B » publie les services qu'elle propose dans l'annuaire à l'aide d'un fichier WSDL. Le client (entreprise « A ») recherche les services disponibles dans l'annuaire (UDDI) puis télécharge les fichiers WSDL depuis ce registre. A partir des informations

récoltées dans les fichiers WSDL, le client peut invoquer le service Web pour obtenir les informations qu'il recherche.

3.4.4.1.2 Description de services

La couche description de service permet de décrire l'interface publique (point d'entrée) du service Web. Le format utilisé est WSDL, ce dernier est un standard XML proposé par W3C pour décrire l'interface d'un service. WSDL inclut les informations suivantes :

- le protocole de communication (SOAP ou RPC par exemple)
- les méthodes que l'utilisateur peut invoquer
- la localisation du service
- le format des messages

3.4.4.1.3 Communication

La couche communication permet de formater les informations échangées entre les deux partenaires de sorte que le message soit compris par le client et par le serveur. Le protocole de communication est SOAP qui est détaillé dans la section « principes de base de SOAP ».

3.4.4.1.4 Transport

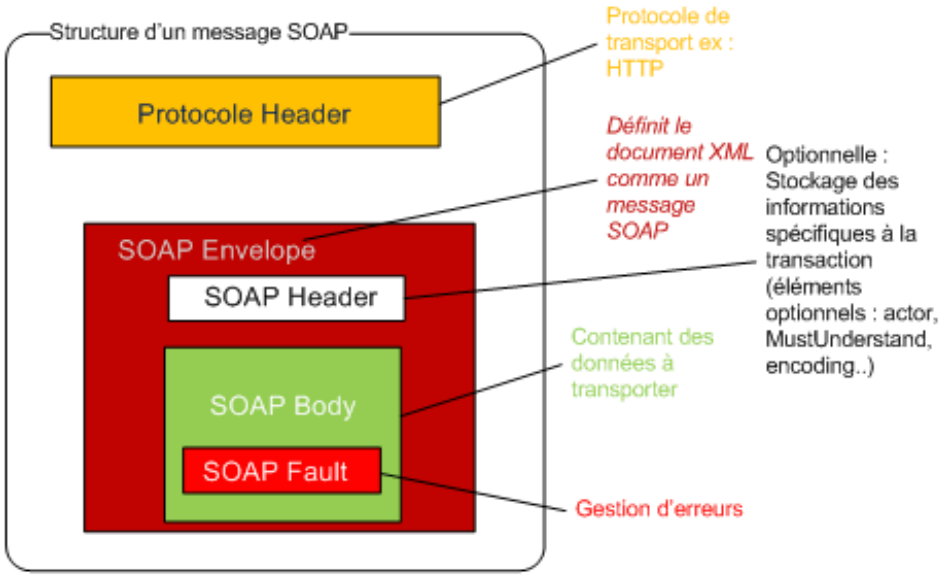
La couche transport est responsable de l'acheminement des messages XML entre le client et le serveur. Comme indiqué précédemment, SOAP peut utiliser plusieurs protocoles de communication (HTTP, FTP, SMTP).

3.4.4.2 Principes de base de SOAP

Le protocole de communication SOAP est constitué en trois parties :

- enveloppe SOAP (SOAP Envelope) qui est obligatoire
- corps de message (SOAP Body) qui est obligatoire
- en-tête SOAP (SOAP Header) qui est optionnel

Figure 24 : L'enveloppe SOAP



Source : REFES, 12 août 2016

L'enveloppe constitue la racine d'un message SOAP. Cette dernière sert de conteneur aux autres éléments (le SOAP Header et le SOAP Body). Le début de l'enveloppe est défini par la balise <soap :Envelope> et se termine avec la balise </soap :Envelope>. Les balises associées à la communication SOAP sont préfixés par « soap », dans ce cadre, la balise concernant l'en-tête est <soap :Header> et la balise concernant le corps du message est <soap :Body>. La figure ci-dessous est un exemple d' enveloppe SOAP.

Figure 25 : Exemple d'une enveloppe SOAP

```
<?xml version="1.0" encoding="utf-8"?>
 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
       soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">

    <soap:Header>
        <!-- en-tête -->
    </soap:Header>

    <soap:Body>
        <!-- corps -->
    </soap:Body>
</soap:Envelope>
```

Source : REFES, 12 août 2016

La spécification SOAP définit deux espaces de noms (namespace) décrits dans le tableau suivant. Ils permettent d'identifier les balises pour éviter des ambiguïtés.

Tableau 6 : Namespace SOAP

Spécification	URI	Détail
xmlns:soap	http://schemas.xmlsoap.org/soap/envelope	Permet de définir l'espace de noms de l'enveloppe
soap:encodingStyle	http://schemas.xmlsoap.org/soap/encoding	Permet de définir le format de types de données

Il est important de noter que l'enveloppe SOAP a la même structure tant pour la requête que pour la réponse.

L'en-tête est une balise facultative qui, lorsqu'elle est présente doit apparaître avant la balise du corps (Body). Cette balise est utilisée pour ajouter des informations supplémentaires à échanger entre les deux partenaires. Dans ce cadre, les utilisateurs peuvent en faire usage comme bon leur semble. Typiquement, cette balise est employée pour fournir des informations d'authentification (section sécurité) ou simplement pour passer des informations intermédiaires.

Le corps SOAP contient toute l'information dont le receveur de la requête a besoin pour répondre à la demande. En d'autres termes, les appels au service Web se font à ce niveau. Le corps fournit alors le nom de la méthode à invoquer ainsi que ses paramètres (s'il y en a). L'exemple ci-dessous décrit un appel et la réponse correspondante d'une opération SOAP sans paramètre.

Figure 26 : Exemple d'un appel SOAP sans paramètre

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <ns2:sayHelloToTheWorld
            xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/"/>
    </soap:Body>
</soap:Envelope>
```

Source : REFES, 12 août 2016

Figure 27 : Exemple d'une réponse à un appel SOAP sans paramètre

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <ns2:sayHelloToTheWorldResponse
            xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/">
            <return>Hello World</return>
        </ns2:sayHelloToTheWorldResponse>
    </soap:Body>
</soap:Envelope>
```

Source : REFES, 12 août 2016

L'exemple ci-dessous décrit un appel et la réponse correspondante d'une opération SOAP avec un paramètre.

Figure 28 : Exemple d'un appel SOAP avec un paramètre

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <ns2:sayHelloToTheWorld
            xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/"/>
            <name>Joao</name>
    </soap:Body>
</soap:Envelope>
```

Source : REFES, 12 août 2016

Figure 29 : Exemple d'une réponse à un appel SOAP avec un paramètre

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <ns2:sayHelloToTheWorldResponse
            xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/">
            <return>Hello World Joao</return>
        </ns2:sayHelloToTheWorldResponse>
    </soap:Body>
</soap:Envelope>
```

Source : REFES, 12 août 2016

A l'intérieur du corps SOAP, il est possible d'intégrer la balise <soap:Fault> pour indiquer un problème lors de la tentative d'appel au service Web. Cette balise n'est pas obligatoire et apparaît uniquement dans l'enveloppe de réponse. Une balise <soap :Fault> peut comporter d'autres balises facultatives :

- la balise « faultcode » contient un code qui indique la nature du problème. Les codes d'erreurs possibles sont les suivants :
    - soap :Server qui indique que l'erreur provient du serveur
    - soap :Client qui indique que le message contient une erreur
    - soap :VersionMismatch qui indique que les versions SOAP utilisées côté client et côté serveur sont différentes
    - soap : MustUnderstand indique qu'il y a un problème dans l'en-tête (header)
- la balise « faultstring » est la traduction du code d'erreur sous forme de texte
- la balise « faultactor » indique d'où l'erreur provient. Cette balise est très intéressante lorsque la chaîne de demande passe par plusieurs services.
- la balise « detail » contient des informations sur l'état du serveur à l'instant de l'apparition de l'erreur.

L'exemple ci-dessous décrit l'utilisation d'une balise <soap :Fault>.

Figure 30 : Exemple d'une balise <soap :Fault>

```
<soap:Body>
    <soap:Fault>
        <!--Identifiant de l'erreur ? défini par SOAP-->
        <faultcode>soap:Server</faultcode>
        <!--Description brève du message-->
        <faultstring>Impossible de router le message.</faultstring>
        <!--Composant qui a généré l'erreur (URL)-->
        <faultactor>http://www.monsite.com/messageDispatcher</faultactor>
        <!--Message spécifique à l'application-->
        <detail>
            <m:error xmlns:m="http://www.monsite.com/errors"> E_NO_ROUTE </m:error>
        </detail>
   </soap:Fault>
</soap:Body>
```

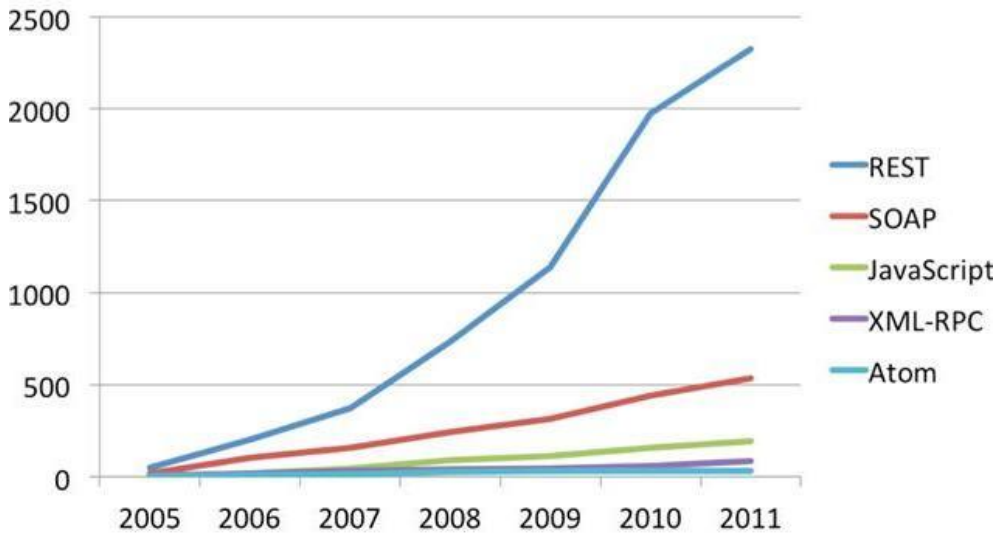
Source : REFES, 12 août 2016

3.4.5 Comparaison

SOAP et REST sont les méthodes les plus utilisées pour échanger des données entre un client et un serveur. Il est difficile de comparer ces deux méthodes, car elles ont des approches différentes. Comme expliqué dans les sections précédentes, REST est un style d'architecture utilisant la technologie du Web tandis que SOAP définit un protocole qui permet d'appeler des procédures à distance. Aucune méthode n'est meilleure que l'autre, tout dépend du service qu'il faut implémenter. En effet, REST est conseillé pour un service Web orienté ressources qui a besoin d'utiliser les méthodes HTTP (CRUD). SOAP est plutôt utilisé lorsque le CRUD ne suffit pas, dès lors que des opérations plus complexes sont nécessaires. En termes de sécurité, les services REST s'appuient sur la sécurité gérée par le protocole HTTP tant pour l'authentification que pour le cryptage (HTTPS). SOAP offre en plus une sécurité intégrée (WS-Security).

Les applications mobiles et les applications Web ont permis à REST de prendre l'avantage sur SOAP dès 2006, car cette architecture est plus adaptée aux clients d'aujourd'hui (smartphone, tablette). Depuis l'utilisation de REST a augmenté de façon spectaculaire et ne montre aucun signe de ralentissement tandis que le nombre d'implémentations en SOAP est en faible croissance.

Figure 31 : Evolution de REST vs SOAP



Source : EVANS, 30 juin 2016

Le tableau suivant est un résumé des caractéristiques principales de REST et SOAP.

Tableau 7 : Comparaison REST vs SOAP

REST	**SOAP**
Style architectural	Protocole
Utilise XML ou JSON pour envoyer et recevoir des informations	Utilise WSDL pour communiquer entre le client et le serveur
Appelle le service via une URL	Utilise des appels de méthodes RPC
Uniquement le protocole HTTP	Principalement HTTP, mais peut utiliser SMTP, FTP, …
Sécurité s'appuyant sur celle du protocole HTTP	Sécurité intégrée via « WS-Security » en plus de celle du protocole

Facile à mettre en place	Complexité, lourdeur de la mise en place et de la maintenance

3.5 Sécurité des services Web

Les technologies des services Web sont des solutions éprouvées pour répondre aux besoins actuels :

- interconnexion des SI
- croissance des services en lignes
- nécessiter de communiquer des informations de l'entreprise vers l'extérieur au moyen d'Internet

Mais la réponse aux besoins ci-dessus engendre un défi majeur, celui de la sécurité. Dès lors qu'une entreprise envisage de créer un service Web, elle doit répondre aux trois questions suivantes pour définir le niveau de sécurité nécessaire :

- quelle est la probabilité que le service soit attaqué ?
- quels sont les risques/conséquences si les données sont récupérées par une personne à mauvais escient ?
- quel est le niveau de sensibilité des données exposées ?

Car en mettant en place un service Web, une entreprise expose ses données et donc son métier vers l'extérieur, mais en générale ces données ne doivent pas être accessibles à des utilisateurs n'ayant pas les autorisations adéquates. Des techniques côté réseau et hardware ainsi que des techniques au niveau de l'architecture choisie (REST ou SOAP) permettent de rendre un service Web sécurisé.

3.5.1 Sécurité côté réseau et hardware

Un problème de sécurité peut advenir lorsqu'un tiers se place entre deux machines et intercepte les échanges de données entre ces dernières. Ce type d'attaque est appelé attaque de l'homme du milieu (HDM). Les solutions à ce type d'attaque sont les suivantes :

- utilisation du protocole HTTPS. Ce protocole est une combinaison du protocole HTTP avec une couche de chiffrement notamment le SSL/TLS. Il permet ainsi aux utilisateurs de consulter des données sensibles (comptes bancaires) sur Internet de façon sécurisée, car les données qui transitent entre l'utilisateur et le serveur sont cryptées.
- utilisation d'un VPN. Le VPN est un réseau privé virtuel se comportant comme un réseau local permettant de créer un lien direct (tunnel) entre deux machines distantes.

Les deux solutions ci-dessus permettent de garantir la confidentialité de l'échange entre deux machines distantes.

Pour limiter les appels de services Web, il est possible de mettre en place un filtrage d'adresse MAC ou d'adresse IP. Cependant, pour ce dernier, le filtrage peut être facilement détourné par un masquage d'adresse via des logiciels tiers. Par conséquent, le filtrage IP est une solution de sécurité intéressante, mais insuffisante si elle est utilisée seule.

3.5.2 Sécurité en REST

3.5.2.1 Authentification basique HTTP

L'authentification HTTP permet à un utilisateur de s'authentifier auprès d'un serveur HTTP pour accéder aux ressources à accès restreint de ce dernier. La restriction des accès est généralement définie dans le fichier de configuration serveur fournissant le service HTTP. Cette authentification est la solution la plus facile à mettre en œuvre côté développement, car la plupart du temps, elle peut être utilisée sans utiliser des bibliothèques supplémentaires. En général, les langages de programmation proposent ce service de base sans aucune installation supplémentaire. Le problème de l'authentification de base est comme son nom l'indique trop basique, cette solution ne propose pas d'options avancées pour son utilisation parce qu'elle requiert uniquement un nom d'utilisateur et un mot de passe. L'authentification est ensuite codée en Base64 qui a le désavantage d'être facilement décodable par un logiciel ou même un humain.

Le codage Base64 permet de représenter une suite de bit quelconque en utilisant uniquement 65 caractères soit les caractères suivants :

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=

Pour l'exemple ci-dessous, le nom d'utilisateur est « joao » et le mot de passe est « cscqheg » soit « joao:cscqheg » selon la syntaxe de l'authentification basique. Chaque caractère (char) correspond à un octet soit 8 bits. Par la suite, à l'aide de la table ASCII[2], il faut récupérer la valeur décimale de chaque caractère de la chaîne « joao:cscqheg ». Il faut ensuite convertir chaque valeur décimale afin d'obtenir la valeur binaire.

Tableau 8 : Convertir une chaîne de caractère en binaire

Nom d'utilisateur		
Caractère	**Décimal**	**Binaire**
j	106	01101010

[2] Annexe 1

o	111	01101111
a	97	01100001
o	111	01101111
Séparateur		
Caractère	**Décimal**	**Binaire**
:	58	00111010
Mot de passe		
Caractère	**Décimal**	**Binaire**
c	99	01100011
s	115	01110011
c	99	01100011
q	113	01110001
h	104	01101000
e	101	01100101
g	103	01100111

Donc la suite de bits de ce nom d'utilisateur et de ce mot de passe est :

01101010 | 01101111 | 01100001 | 01101111 | 00111010 | 01100011 | 01110011 | 01100011 | 01110001 | 01101000 | 01100101 | 01100111

Le fonctionnement du codage Base64 est de transformer 3 octets de 24 bits en 4 paquets de 6 bits (4x6 = 24 bits) et de répéter l'opération pour chaque groupe de trois caractères, soit :

011010 | 100110 | 111101 | 100001 | 011011 | 110011 | 101001 | 100011 | 011100 | 110110 | 001101 | 110001 | 011010 | 000110 | 010101 | 100111

Cependant, il se peut que le dernier groupe de trois caractères ne soit pas complet (3 octets), car il peut être composé de 1 ou 2 octets. Si c'est le cas, il faut remplir les caractères manquant avec la valeur 0. Si le dernier groupe de trois caractères n'est composé que d'un seul caractère (octet) alors seuls les 3 premiers paquets de 6 bits sont utilisés et le dernier paquet est complété par le caractère « = ». Si le dernier groupe de trois caractères n'est composé que de deux caractères (2 octets) alors seuls les deux premiers paquets de 6 bits sont utilisés et les deux derniers paquets sont complétés par le caractère « = ».

Par la suite, il faut faire correspondre la nouvelle séquence de bits aux valeurs de l'alphabet Base64 [3]soit :

Tableau 9 : Convertir une séquence de bits en valeur Base64

Bits	**Valeur alphabet Base64**
011010	a
100110	m
111101	9
100001	h
011011	b
110011	z
101001	p
100011	j
011100	c
110110	2
001101	N
110001	x

[3] Annexe 2

011010	a
000110	G
010101	V
100111	n

Par conséquent, la chaîne « joao:cscqheg » codée en Base64 donne le résultat « am9hbzpjc2NxaGVn ». Pour décoder une chaîne, il faut réaliser les mêmes manipulations dans le chemin inverse.

Cet exemple démontre qu'il est très facile pour un tiers qui se place au milieu de la communication d'intercepter et de décoder l'échange. C'est pour cette raison que l'authentification basique doit être utilisée uniquement avec le protocole HTTPS qui permet de crypter la communication. A contrario, il est fortement déconseillé d'utiliser l'authentification de base avec le protocole HTTP parce que la combinaison de nom d'utilisateur et mot de passe peut être facilement décodée.

3.5.2.2 OAuth

OAuth est un standard ouvert pour l'autorisation et non un protocole d'authentification. La première version (OAuth1.0a) est standardisée depuis avril 2010 et sa version actuelle (OAuth2.0) est disponible depuis octobre 2012. Cette dernière version corrige des problèmes de sécurité, mais simplifie également l'interopérabilité entre les différentes applications. Dans ce cadre, OAuth permet à une application d'utiliser l'API sécurisée d'un autre fournisseur pour authentifier un utilisateur. A cet effet, l'application implémente le service OAuth mis à disposition par le fournisseur. L'authentification est alors gérée par le fournisseur qui retourne à l'application l'autorisation ou le refus de connexion et éventuellement d'autres informations.

Le principal avantage d'utiliser ce standard est que l'utilisateur n'est pas obligé de créer un compte spécifique pour l'application fournissant le service.

Les sites Internet et les applications mobiles proposent de plus en plus à un utilisateur de se connecter à partir de son compte Twitter, Facebook ou encore Google. Cette fonctionnalité a l'avantage de faciliter les connexions de l'utilisateur à l'application, mais a pour désavantage de potentiellement fournir des informations relatives à votre compte. Par exemple, l'API OAuth de Google retourne non seulement l'autorisation de connexion, mais également des informations sur profil de l'utilisateur telles que le nom

complet ou l'adresse email. Il est évident que l'utilisateur doit faire confiance à l'application faisant appel au service OAuth d'un fournisseur tiers avant une éventuelle connexion.

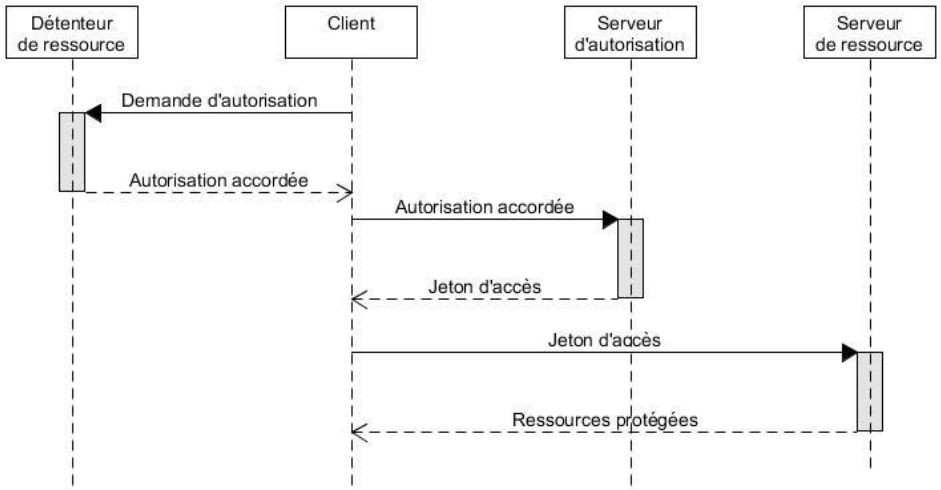
Côté développement beaucoup de bibliothèques ont été conçues dans les langages de programmation les plus utilisés pour permettre d'implémenter cette technologie dans les nouvelles applications.

OAuth est caractérisé par quatre rôles distincts :

- le détenteur des données (Ressource Owner) est l'entité qui accorde ou non l'accès à la ressource protégée
- le serveur de ressources (Ressource Server) héberge les données (ressources) protégées. Ce serveur permet de valider la demande du client en utilisant les jetons (tokens) afin de répondre à sa requête.
- le client (Client) est l'application qui demande les données (application mobile, site Internet, …)
- le serveur d'autorisation (Authorization Server) permet de délivrer des jetons (tokens) au client après avoir obtenu l'autorisation auprès du propriétaire de la ressource. Les jetons (tokens) sont utilisés dès lors que le client envoie une requête vers le serveur de ressources. A noter que ce serveur peut être sur la même machine que le serveur de ressources.

La figure ci-dessous permet de mieux comprendre les échanges entre les différents rôles de ce protocole.

Figure 32 : Diagramme de séquence OAuth



Source : RATSITOBAINA, 11 août 2016

Le jeton (token) est une chaîne de caractère générée par le serveur d'autorisation à la demande du client. Il existe deux types de jetons :

- le jeton d'accès (Access Token) a une durée d'accès limité qui est définie par le serveur d'autorisation. Il permet au client de récupérer les données (ressources) protégées sur le serveur de ressource. Chaque requête vers le serveur de ressource doit contenir le jeton en tant que paramètre ou dissimulé à l'intérieur de l'en-tête (header).
- le jeton de renouvellement (Refresh Token) n'est pas envoyé au serveur de ressource à chaque requête, il permet de renouveler le jeton d'accès lorsque ce dernier est expiré. Dans ce cadre, le client demande un nouveau jeton d'accès au serveur d'autorisation.

OAuth introduit la notion du « scope ». Un scope permet de définir les droits d'un jeton d'accès. La liste des « scopes » est définie par le serveur d'autorisation. Quand le client demande une autorisation auprès du serveur d'autorisation, il doit également préciser les « scopes » qu'il souhaite utiliser.

Concernant le protocole de communication, OAuth impose l'utilisation du protocole HTTPS, car des données sensibles transitent entre le client et le serveur notamment le jeton (token).

3.5.3 Sécurité en SOAP

3.5.3.1 WS-Security

La norme WS-Security propose différentes possibilités pour sécuriser des services Web de type SOAP. Des entreprises comme IBM ou encore Microsoft sont à l'origine de cette norme qui est maintenant développée et maintenue par Oasis-Open. La première version (WS-Security 1.0) a été lancée le 19 avril 2004 et sa version actuelle (WS-Security 1.1) a été lancée le 17 avril 2006.

Les spécifications WS-Security expliquent la façon dont la confidentialité et l'intégrité peuvent être appliquées aux services Web qui sont confrontés à utiliser des données sensibles. Cette norme permet donc aux services Web de sécuriser l'information et de la rendre accessible uniquement aux personnes ou systèmes habilités à l'utiliser. A titre informatif, WS-Security est une spécification compliquée à mettre en place, c'est pour cette raison que des frameworks (CXF, Métro, JBossWS, …) ont été développés pour faciliter son implémentation.

Il est important de noter que cette norme ne s'applique qu'au niveau du message envoyé entre le client et le serveur et qu'elle n'a pas d'impact sur la couche transport. Comme pour l'architecture REST, les services SOAP doivent passer par le protocole HTTPS ou par un VPN pour avoir une communication cryptée.

En utilisant les spécifications WS-Security, un service Web peut incorporer les mécanismes suivants pour un message SOAP :

- l'authentification qui permet d'identifier l'utilisateur qui demande le service.
- la signature qui permet d'assurer l'intégrité du message. Eviter que ce dernier soit modifié pendant l'échange entre le client et le serveur.
- le chiffrement qui permet de crypter le contenu du message.

Dans ce cadre, un message SOAP contient plusieurs en-têtes (headers) pour transmettre les informations liées à la sécurité.

Un des en-têtes est l'authentification qui permet de valider un utilisateur. Pour ce faire, il existe plusieurs méthodes pour s'authentifier dont les plus répandues sont :

- Username Token

  L'une des façons les plus courantes de passer des informations d'authentification est d'utiliser une combinaison nom d'utilisateur et mot de passe. Cette méthode ressemble beaucoup à l'authentification basique vue précédemment. Il est possible de passer les informations d'authentification en clair.

Figure 33 : Exemple UsernameToken non sécurisé

```
<wsse:UsernameToken>
    <wsse:Username>scott</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">password</wsse:Password>
</wsse:UsernameToken>
```

Source : SEELY, 12 août 2016

Cette façon de faire est déconseillée, car le mot de passe est de type «wsse :PasswordText », ce qui signifie que le mot de passe transite en clair. Mais il est possible d'envoyer les informations d'authentification de manière plus sûre grâce au type «wsse :PasswordDigest ». Cela ajoute plus de sécurité, car le mot de passe est transformé par une fonction de hachage en SHA1. Le mot de passe transformé est une concaténation du « nonce », de « l'heure de création » et du « mot de passe ». Le paramètre « nonce » est une chaîne de 16 caractères codés en Base64. La figure ci-dessous est un exemple d'une authentification plus sûre via le type «wsse :PasswordDigest » :

Figure 34 : Exemple UsernameTocken sécurisé

```
<wsse:UsernameToken>
    <wsse:Username>scott</wsse:Username>
    <wsse:Password Type="wsse:PasswordDigest">
        KE6QugOpkPyT3Eo0SEgT30W4Keg=</wsse:Password>
    <wsse:Nonce>5uW4ABku/m6/S5rnE+L7vg==</wsse:Nonce>
    <wsu:Created xmlns:wsu=
        "http://schemas.xmlsoap.org/ws/2002/07/utility">
            2002-08-19T00:44:02Z
    </wsu:Created>
</wsse:UsernameToken>
```

Lorsque le client reçoit le message, il doit générer un mot de passe du même type avec les mêmes informations pour pouvoir comparer les deux mots de passe transformés (celui qu'il a reçu de l'en-tête et celui qu'il a généré). Si les deux mots de passe concordent alors l'authentification est validée. L'inconvénient de cette protection est qu'elle ne protège pas les attaques répétitives, il faut donc penser à mettre un délai d'expiration et lorsque ce délai est dépassé le mot de passe transformé doit être régénéré.

- X.509 certificat

  Une autre option pour s'authentifier est d'utiliser un certificat X.509. Dans le message SOAP, il est possible d'intégrer un certificat dans l'en-tête selon la norme X.509. Ce certificat permet au serveur d'identifier l'émetteur du message et inclut une clé publique. Il est ainsi possible de vérifier si l'utilisateur est autorisé à demander un service. Cependant, un utilisateur non autorisé pourrait inclure le certificat d'un autre utilisateur autorisé dans son message. Aussi il est demandé que l'émetteur du message crée une signature digitale du message avec sa clé privé. Cette signature pourra être vérifiée en utilisant la clé publique du certificat. Dès lors un jeton (BinarySecurityToken) est créé et représenté sous le format suivant.

Figure 35 : Exemple X.509 certificat

```
<wsse:BinarySecurityToken
    ValueType="wsse:X509v3"
    EncodingType="wsse:Base64Binary"
    Id="SecurityToken-f49bd662-59a0-401a-ab23-1aa12764184f"
>MIIHdjCCB...</wsse:BinarySecurityToken>
```

  - L'information « ValueType » est X509v3 (Certificat X.509 v3).

- L'information « EncodingType » précise le type d'encodage :
  - Base64Binary
  - HexBinary
- L'information « Id » est un identifiant unique.

Pour éviter toute sorte de combinaison d'authentification ou d'attaques, il est important de penser à ajouter le paramètre « wsu:Timestamp» dans l'en-tête SOAP pour donner un délai de vie au message avant qu'il ne soit ignoré.

La signature du message indique au destinataire que le message n'a pas été falsifié pendant la communication. Le message est signé à l'aide du XML Signature (recommandé par W3C). Cette technique permet d'avoir des signatures numériques dans les documents XML. WS-Security explique comment utiliser cette signature avec les mécanismes d'authentification comme « Username Token » ou « X.509 certificat » par exemple. En utilisant l'authentification « Username Token », l'expéditeur peut signer le message à l'aide du mot de passe. Avec l'authentification « X.509 certificat », l'expéditeur signe le message avec la clé privé du certificat.

Dans certains cas, il se peut que l'authentification et la signature ne suffisent pas. En effet, si le numéro de carte de crédit est envoyé en texte brut et de manière signée, une personne qui se place au milieu de la communication peut constater après plusieurs échanges que les données qui transitent sont réelles et valides (elles n'ont pas changé). Par conséquent, WS-Security spécifie comment crypter les données via XML Encryption de manière à ce que seul le destinataire puisse lire le message. XML Encryption est spécifié et recommandé par W3C qui permet de crypter des documents XML. Le cryptage des données peut être :

- symétrique qui permet aux deux parties (client et serveur) qui dialoguent d'avoir une clé connue par elles seules. Mais cette manière de faire pose le problème de l'échange de la clé. Il est conseillé d'utiliser cette méthode que si un contrôle des deux partenaires est géré et qu'une confiance règne avec les clients ou les applications qui utilisent les clés.
- asymétrique qui utilise une clé publique pour le cryptage et une clé privée pour le décryptage.

La figure ci-dessous montre un exemple de message SOAP crypté.

Figure 36 : Exemple enveloppe SOAP cryptée

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <soap:Header
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
        xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
        <wsu:Timestamp>
            <wsu:Created
                wsu:Id="Id-3beeb885-16a4-4b65-b14c-0cfe6ad26800"
                >2002-08-22T00:26:15Z</wsu:Created>
            <wsu:Expires
                wsu:Id="Id-10c46143-cb53-4a8e-9e83-ef374e40aa54"
                >2002-08-22T00:31:15Z</wsu:Expires>
        </wsu:Timestamp>
        <wsse:Security soap:mustUnderstand="1" >
            <xenc:ReferenceList>
                <xenc:DataReference
        URI="#EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51" />
            </xenc:ReferenceList>
            <xenc:ReferenceList>
                <xenc:DataReference
        URI="#EncryptedContent-666b184a-a388-46cc-a9e3-06583b9d43b6" />
            </xenc:ReferenceList>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <xenc:EncryptedData
            Id="EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51"
            Type="http://www.w3.org/2001/04/xmlenc#Content">
            <xenc:EncryptionMethod Algorithm=
                "http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                <KeyName>Symmetric Key</KeyName>
            </KeyInfo>
            <xenc:CipherData>
                <xenc:CipherValue
                >InmSSXQcBV5UiT...  Y7RVZQqnPpZYMg==</xenc:CipherValue>
            </xenc:CipherData>
        </xenc:EncryptedData>
    </soap:Body>
</soap:Envelope>
```

Source : SEELY, 12 août 2016

Le message SOAP contient les informations sur quelles données le cryptage a été réalisé et de quelle façon (symétrique ou asymétrique). Pour toutes les personnes qui ne connaissant pas la clé, le texte chiffré à l'intérieur du corps du message (soap :Body) ne peut pas être déchiffré. Dans le cas d'un cryptage asymétrique, le récepteur du message doit connaître la clé privée afin de déchiffrer et lire le message.

Il est important de noter que WS-Security peut être complété avec d'autres extensions « WS-* » pour augment le niveau de sécurité. Notamment des spécifications tel que :

- WS-Policy
- WS-Privacy

3.6 Format de sortie de données

Le format de sortie de données est un élément important dans la construction d'un service Web, car les appareils mobiles doivent pouvoir facilement consommer ce service. De nos jours, les périphériques mobiles sont pour la plupart liés à un plafond

mensuel de données et selon le contrat, la vitesse de transmission peut être limitée. C'est pour cette raison qu'un format de donnée doit être aussi léger et efficace que possible.

3.6.1 HTML

Un des formats de données les plus connus est HTML, celui-ci est conçu pour afficher les pages Web, mais peut également être utilisé comme format de sortie d'un service Web. Utiliser HTML peut rapidement devenir un problème avec un service Web, car ce format de données est lourd, il contient des balises qui n'ont aucune importance pour les consommateurs de données pures (client de services Web). Un tel format est contraignant pour les utilisateurs limités en ressources et en vitesse.

Un service Web propose très rarement un format de sortie HTML, les formats de retour les plus utilisés sont aujourd'hui XML et JSON qui proposent des performances nettement meilleures.

3.6.2 XML

XML est un métalangage permettant d'organiser des données à l'aide de balises. Il est très similaire au HTML à la différence qu'en XML, il est possible de définir des balises personnalisées.

Une balise est identifiée par un nom décrivant généralement son contenu. Le début du contenu est marqué par la séquence < nom de la balise >. La fin du contenu est marquée par la séquence < / nom de la balise >. Un exemple simple de fichier XML contenant deux balises (nom et ecole) est présenté sur la figure suivante.

Figure 37 : Exemple simple XML

```
<nom>João Diogo Amaral Rodrigues</nom>
<ecole>Haute Ecole de gestion de Genève</ecole>
```

La balise <nom> indique le début de l'information et la balise </nom> indique la fin de l'information. Le contenu (João Diogo Amaral Rodrigues) est défini entre ces deux balises. Un fichier XML est un moyen commode d'échange d'informations entre différents logiciels ou systèmes d'exploitation. En effet, il est sous forme de texte (non binaire) et les balises qu'il contient permettent de lire ou récupérer facilement les informations contenues.

XML permet de définir une structure hiérarchique des informations contenues dans le fichier. En effet, une balise peut contenir d'autres balises, ce qui permet d'imbriquer des

informations à l'infini. L'exemple ci-dessous montre une collection de trois d'étudiant. Chaque étudiant de cette collection est associé à un nom et une école.

Figure 38 : Exemple d'une structure hiérarchique en XML

```
<etudiants>
    <etudiant>
        <nom>João Diogo Amaral Rodrigues</nom>
        <ecole>Haute Ecole de gestion de Genève</ecole>
    </etudiant>
    <etudiant>
        <nom>Christophe Dier</nom>
        <ecole>Haute Ecole de santé de Genève</ecole>
    </etudiant>
    <etudiant>
        <nom>Luis Filipe</nom>
        <ecole>Ecole de Commerce Aimée-Stitelmann</ecole>
    </etudiant>
</etudiants>
```

Il existe de nombreuses spécifications ou recommandations qui permettent de manipuler du XML telles que :

- XSD : permet de définir une structure précise du fichier XML, cet outil détermine les balises obligatoires, le type d'information que contient une balise (String, int) et qu'elle balise contient d'autres balises.
- XSLT : permet d'automatiser les transformations d'un fichier XML vers d'autre format tel que HTML ou encore CSV.
- XQuery : permet d'extraire de l'information d'un fichier XML, c'est l'équivalent d'une requête en SQL.

3.6.3 JSON

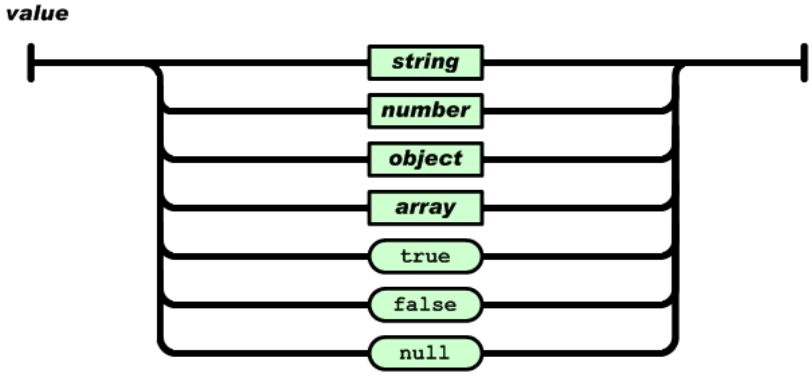
JSON est un format léger de données qui permet de stocker et d'échanger des informations. Tout comme son concurrent direct (XML), un fichier JSON est également au format texte. Cependant, les utilisateurs de ce format ont une légère préférence pour le JSON, car ils estiment qu'il est plus facile à lire et à écrire. JSON n'est pas un langage de programmation même si les conventions qu'il utilise sont dérivées du langage « C » (Java, C#, JavaScript, …).

JSON est construit sur deux structures :

- des ensembles de couples nom/valeur (l'équivalant d'un objet ou d'un enregistrement pour un langage de programmation)
- une liste de valeurs ordonnées (l'équivalant d'un tableau ou d'un vecteur pour un langage de programmation)

Une valeur peut avoir l'un des types décrits dans la figure suivante.

Figure 39 : Représentation d'une valeur en JSON



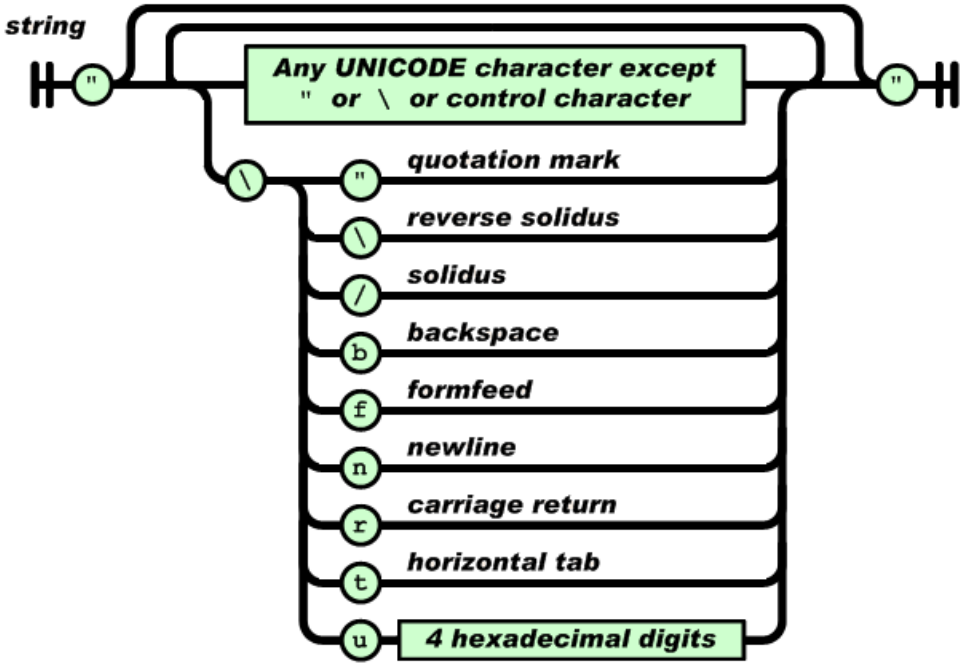
Source : JSON ORG, 4 juillet 2016

Ces éléments structurels permettent de représenter différents types de données :

- une chaîne de caractères

  Exemple d'une chaîne de caractères : "joao"

Figure 40 : Représentation d'une chaîne de caractères en JSON

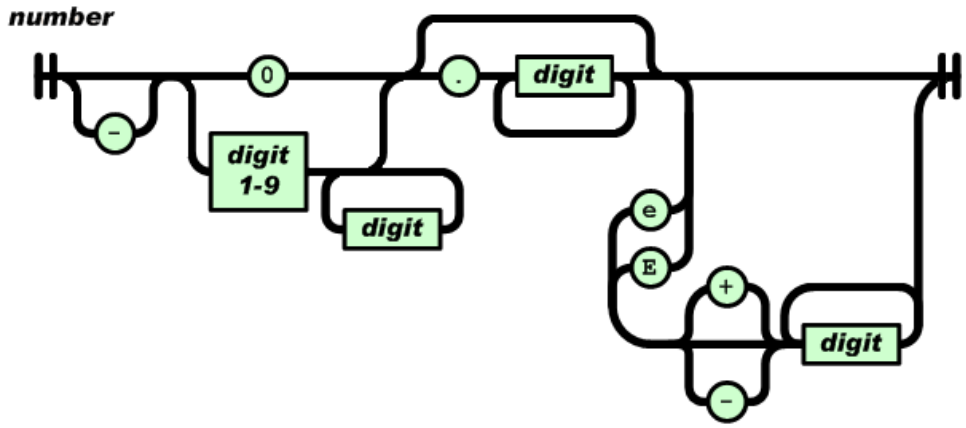


Source : JSON ORG, 4 juillet 2016

- un nombre

  Exemple d'un nombre : 10

Figure 41 : Représentation d'un nombre en JSON

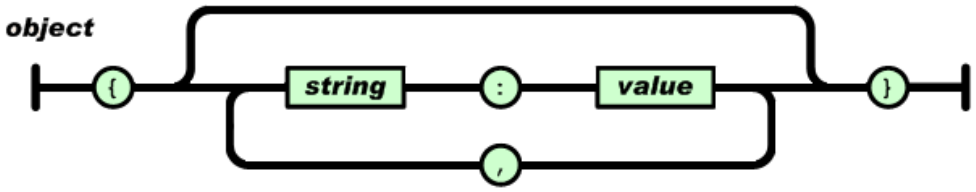


Source : JSON ORG, 4 juillet 2016

- un objet

  Un objet se caractérise par un ensemble de couples nom/valeur. Le nom et la valeur sont séparés par deux points tandis que les ensembles sont séparés par une virgule. Exemple d'un objet « personne »: {"nom":"João Diogo Amaral Rodrigues", "ecole":"Haute Ecole de gestion de Genève"}.

Figure 42 : Représentation d'un objet en JSON

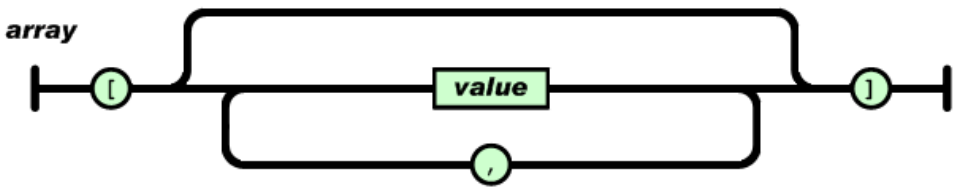


Source : JSON ORG, 4 juillet 2016

- un tableau

  Le tableau représente une collection. Les valeurs sont séparées par une virgule. Exemple d'un tableau d'entier : [2016, 2020, 2024, 2028].

Figure 43 : Représentation d'un tableau en JSON



Source : JSON ORG, 4 juillet 2016

Comme pour XML, l'exemple ci-dessous représente une collection d'étudiants contenant trois objets. Chaque objet a deux valeurs, le nom et l'école.

Figure 44 : Exemple d'un tableau d'étudiants en JSON

```json
{
    "etudiants": [
        {
            "nom": "João Diogo Amaral Rodrigues",
            "ecole": "Haute Ecole de gestion de Genève"
        },
        {
            "nom": "Christophe Dier",
            "ecole": "Haute Ecole de santé de Genève"
        },
        {
            "nom": "Luis Filipe",
            "ecole": "Ecole de Commerce Aimée-Stitelmann"
        }
    ]
}
```

Des bibliothèques ont évidemment été développées pour utiliser le format JSON dans les langages de programmation les plus utilisés.

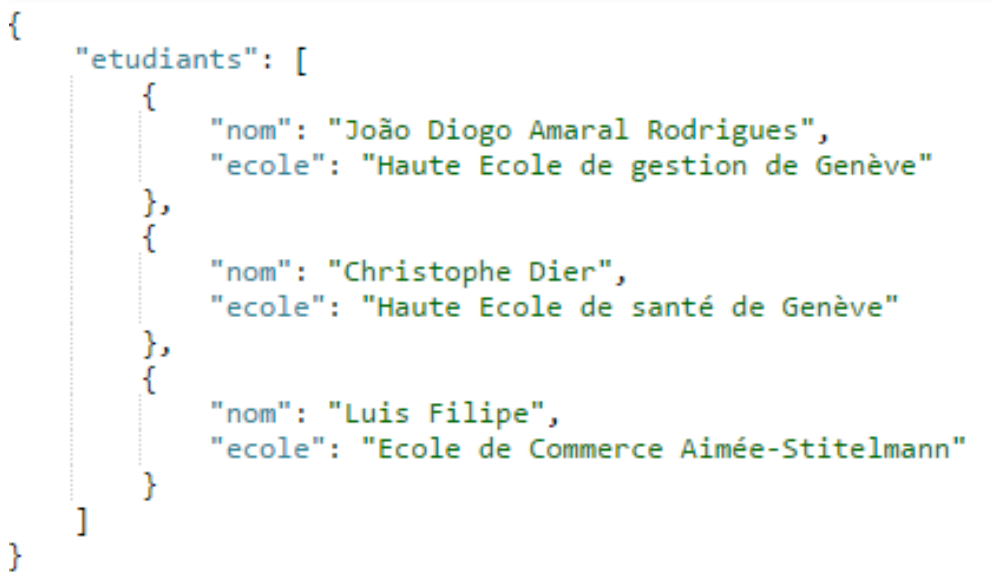
3.6.4 Comparaison

XML et JSON sont les formats d'échange de données les plus utilisés dans le développement d'applications Web. Cependant, JSON est plus léger que XML, car il ne contient pas les balises de début et de fin.

Figure 45 : Fichier XML

```xml
<etudiants>
    <etudiant>
        <nom>João Diogo Amaral Rodrigues</nom>
        <ecole>Haute Ecole de gestion de Genève</ecole>
    </etudiant>
    <etudiant>
        <nom>Christophe Dier</nom>
        <ecole>Haute Ecole de santé de Genève</ecole>
    </etudiant>
    <etudiant>
        <nom>Luis Filipe</nom>
        <ecole>Ecole de Commerce Aimée-Stitelmann</ecole>
    </etudiant>
</etudiants>
```

Figure 46 : Fichier JSON



```
{
    "etudiants": [
        {
            "nom": "João Diogo Amaral Rodrigues",
            "ecole": "Haute Ecole de gestion de Genève"
        },
        {
            "nom": "Christophe Dier",
            "ecole": "Haute Ecole de santé de Genève"
        },
        {
            "nom": "Luis Filipe",
            "ecole": "Ecole de Commerce Aimée-Stitelmann"
        }
    ]
}
```

Dans ce cadre pour une information identique, il y a 313 caractères pour le format XML (figure 45) et 237 caractères pour le format JSON (figure 46) soit une différence de 76 caractères (24%).

4. Pratique

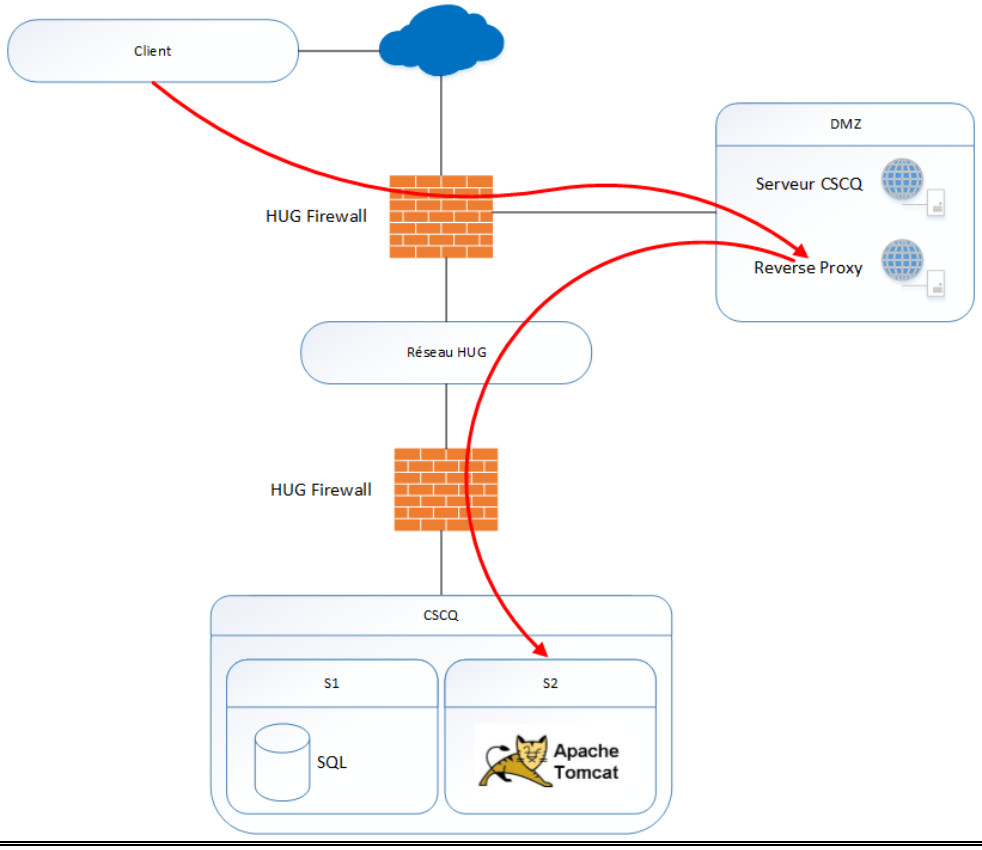
4.1 Architecture et schéma réseau

L'architecture de la solution doit permettre d'accéder depuis l'extérieur (Internet) à des informations stockées dans la base de données interne du CSCQ via un service Web. Une communication de l'extérieur vers le CSCQ n'était pas possible avant ce projet, car les HUG bloquaient les échanges pour des raisons de sécurité du réseau hospitalier. Comme expliqué au début du document, le CSCQ dispose d'une configuration réseau particulière, car l'entreprise est logée dans les locaux des HUG et utilise le réseau informatique de l'hôpital.

Par mesure de confidentialité, le serveur contenant la base donnée interne du CSCQ est nommé « S1 » et la nouvelle machine (nécessaire pour la solution retenue) est nommée « S2 ». Cette dernière dispose d'un serveur « Apache Tomcat » qui gère le service Web du CSCQ.

Pour mener à bien ce projet, plusieurs solutions ont été étudiées en interne (CSCQ) et proposées par la suite aux responsables réseau des HUG. Plusieurs échanges entre les deux parties ont permis de définir une solution satisfaisant à la fois les contraintes du CSCQ et des HUG. La configuration retenue est décrite sur la figure suivante.

Figure 47 : Architecture réseau après l'implémentation de la solution

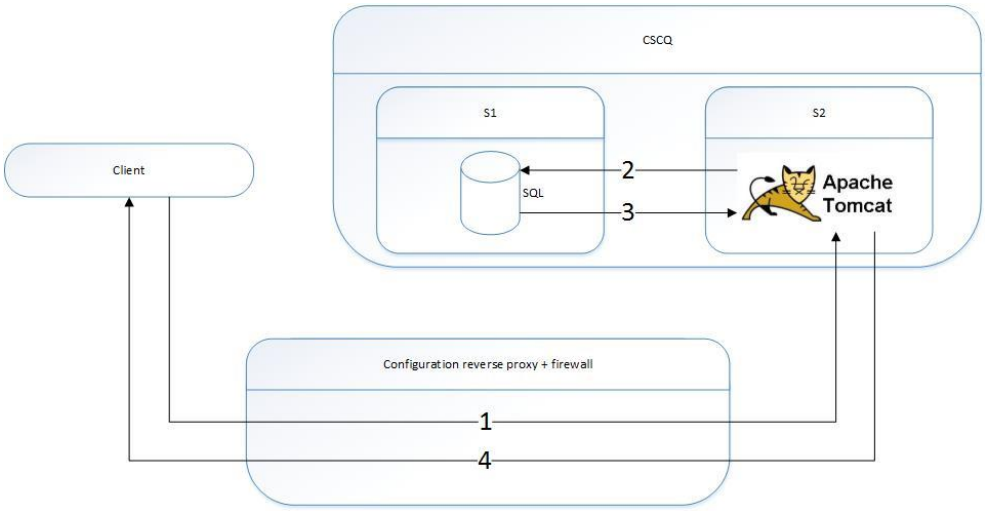


La machine « S2 » est intégrée dans le réseau du CSCQ, mais elle est rendue accessible par Internet. Les accès depuis l'extérieur (Internet) sont filtrés par un premier pare-feu puis traversent un « reverse proxy ». Comme son nom l'indique, un « reverse proxy » est le miroir d'un « proxy ». Un « proxy » permet à un utilisateur d'un réseau interne d'accéder à Internet tandis qu'un « reverse proxy » permet à un utilisateur Internet d'accéder à un réseau interne. Les accès traversent ensuite un second pare-feu puis accèdent à la machine « S2 ».

Pour plus de sécurité, une restriction a été imposée, la machine « S2 » n'est accessible depuis Internet que par le port du serveur « Apache Tomcat ». De plus, les HUG proposent un certificat SSL/TLS, ce qui permet d'utiliser le protocole HTTPS sur la machine « S2 » et donc d'avoir une communication cryptée avec le client.

La prochaine figure permet de mieux comprendre les interactions entre le client et le service Web hébergé sur le serveur « Apache Tomcat » de la machine « S2 ». Cette figure est simplifiée afin de faciliter sa compréhension.

Figure 48 : Interaction entre le client et le service Web



1. Le client fait une requête au service Web hébergé sur le serveur « Apache Tomcat » de la machine « S2 ».
2. Le service Web va récupérer les informations de la requête dans la base de données du CSCQ.
3. La base de données retourne le résultat de la requête au service Web.
4. Le service Web transmet le résultat de la requête au client.

4.2 Tomcat

Tomcat est l'assemblage d'un serveur HTTP et d'un conteneur de servlets. Une servlet est une application JavaEE qui fonctionne côté serveur. Dans ce cadre, une servlet est capable de recevoir et traiter une requête HTTP. JavaEE est en quelque sorte une

évolution de la plateforme standard JavaSE. Cette extension est principalement utilisée par les entreprises et elle propose plus de bibliothèques afin de faciliter le développement d'application Web (servlet). En résumé, Tomcat permet de faire tourner des applications Java côté serveur.

4.3 Accès aux informations (Base de données)

Pour répondre aux requêtes des clients, le service Web qui est hébergé dans la machine « S2 » récupère les informations dans la base de données SQL qui se trouve sur le serveur « S1 ». La communication entre ces deux machines fonctionne sans aucune configuration particulière, car elles sont toutes les deux localisées dans le même réseau et font partie du même « Active Directory ». La machine « S2 » accède à la base de données grâce à son authentification Windows qui utilise le protocole de sécurité « Kerberos ». Cependant, pour des questions de sécurité et d'accès à la base de données, des rôles spécifiques ont été définis avec les privilèges associés pour les services Web ainsi que des utilisateurs avec des droits étendus.

Les prochaines sections détaillent les tables et les procédures stockées qui ont été créées pour gérer le service Web. Ce service fait les demandes à la base de données via des procédures stockées pour éviter l'injection SQL. Pour augmenter la sécurité, il a été mis en place un système de session. Dès lors qu'un adhérent se connecte au service Web, une session est créée avec un temps d'expiration et un quota de requêtes limité. Ce système est courant dans les applications devant gérer des données sensibles notamment les applications bancaires.

4.3.1 Tables

Pour mener à bien ce projet, il a été nécessaire de créer trois nouvelles tables dans la base de données. Toujours par mesure de confidentialité, les trois tables sont nommées « T1 », « T2 » et « T3 ».

La table « T1 »[4] permet de définir les paramètres généraux de l'application (service Web) :

Tableau 10 : Table T1

Attribut	Description
ID	ID de l'application (service Web)

[4] Annexe3

Nom	Nom de l'application (service Web)
Statut	Statut de l'application : • 0 : service indisponible • 1 : service disponible • 2 : service disponible, mais en cours de fermeture (pas de nouvelles connexions possibles)
TempsExpiration	Indique le temps d'expiration de la session en minutes
Quota	Indique le nombre maximum de requêtes pour une session

La table « T2 »[5] permet de mémoriser les sessions :

Tableau 11 : Table T2

Attribut	Description
ID	ID de la session
IDApplication	ID de l'application (service Web)
IDAdherent	ID de l'adhérent (utilisateur qui demande le service)
DateDebutSession	Date et heure du début de la session
DateExpirationSession	Date et heure d'expiration de la session
DateFinSession	Date et heure de fin de la session (session fermée)
NombreAcces	Nombre de requêtes effectué lors de cette session
Staut	Statut de la session : • 0 : session fermée • 1 : session ouverte

[5] Annexe 4

La table « T3 »[6] permet de tracer les demandes du client au service Web (une sorte de log) :

Tableau 12 : Table T3

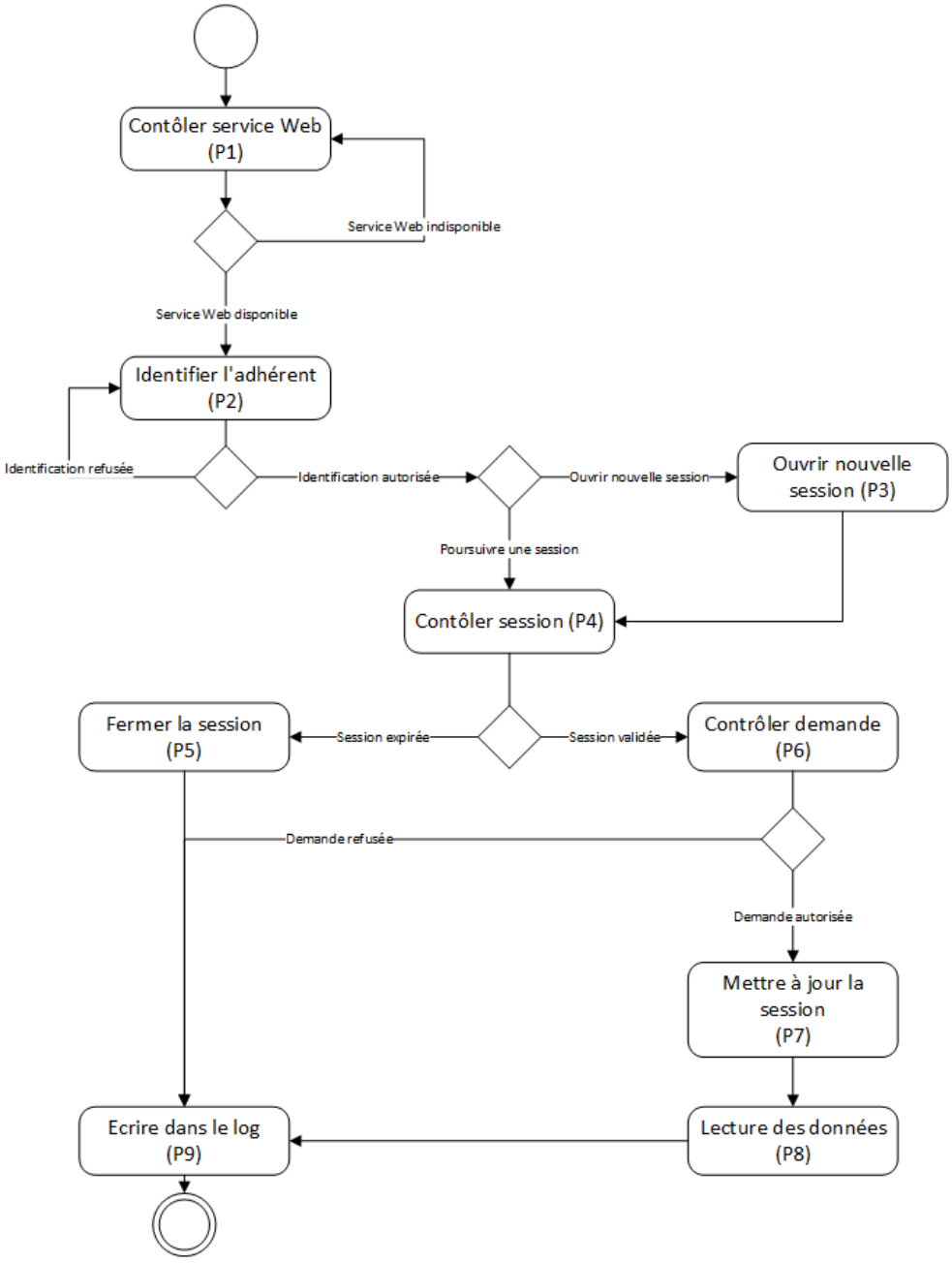
Attribut	Description
ID	ID du traçage (log)
IDSession	ID de la session
ServiceInvoque	Nom du service invoqué (nom de la procédure stockée)
ServiceParametres	Paramètres associés au service invoqué
IDAdherent	ID de l'adhérent
Date	Date et heure de la demande

4.3.2 Procédures stockées

Le diagramme d'activité ci-dessous représente le flux d'un appel au service Web. Ainsi chaque activité de ce flux représente une procédure stockée. Le code des procédures stockées n'est pas fourni, mais le détail de son fonctionnement est décrit.

[6] Annexe 5

Tableau 13 : Diagramme d'activité détaillant les procédures stockées



- La procédure « P1 » : cette procédure permet de savoir si le service Web est disponible ou indisponible. Pour ce faire, celle-ci reçoit en paramètre d'entrée le nom de l'application (service Web), ce qui permet d'aller contrôler dans la table « T1 » via l'attribut « Statut » que le service est disponible ou indisponible. Il y a trois valeurs de retour possible :
    - 0 : le service Web est indisponible (maintenance).
    - 1 : le service Web est disponible. Dans ce cas, l'adhérent peut s'identifier (« P2 »).
    - 2 : le service Web est disponible, mais aucune connexion n'est autorisée (le développeur attend que toutes les sessions soient terminées pour passer le statut à la valeur à 0).

Cette procédure retourne également l'ID de l'application, car cette information est nécessaire pour appeler les autres procédures stockées.

- La procédure « P2 » : cette procédure permet de valider l'authentification de l'adhérent afin de savoir si ce dernier est autorisé à utiliser le service Web. Les paramètres d'entrée sont le nom d'utilisateur et le mot de passe transformé par une fonction de hachage. Si les valeurs d'identification saisies par l'utilisateur coïncident avec les valeurs de la base de données alors la procédure retourne le numéro de l'adhérent et la valeur « O » (oui). Dans le cas contraire, la procédure retourne la valeur « -1 » pour le numéro de l'adhérent et la valeur « N » (non).
- La procédure « P3 » : cette procédure est appelée lorsque le client veut ouvrir une session. Pour arriver à cette étape, le client doit s'authentifier correctement à la procédure « P2 ». La procédure « P3 » reçoit en paramètre l'ID de l'application (service Web) et le numéro de l'adhérent. L'exécution de cette procédure va insérer une nouvelle session dans la table « T2 » (table qui mémorise les sessions). Comme expliqué précédemment une session est caractérisée par les valeurs suivantes :
    - ID : identifiant de la session (incrémentation automatique)
    - IDApplication : valeur fournie en paramètre
    - IDAdherent : valeur fournie en paramètre
    - DateDebutSession : date et heure à laquelle l'adhérent ouvre la session (récupère la date et l'heure du serveur)
    - DateExpirationSession : date et heure à laquelle la session est expirée. Pour calculer cette valeur, il faut récupérer dans la table « T1 » la valeur de l'attribut « TempsExpiration » et l'ajouter à la date et l'heure actuelle. Exemple : la date et heure actuelle = 22.08.2016 08:45:04 et la valeur du temps d'expiration de la table « T1 » = 10 minutes alors la valeur de date d'expiration de cette session = 22.08.2016 08:55:04.
    - DateFinSession : date et heure à laquelle la session a été fermée. A l'insertion cette valeur est « null ».
    - NombreAcces : indique le nombre de requêtes réalisées au cours de la session. A l'insertion cette valeur est égale à 0.
    - Statut : indique si la session est ouverte ou fermée. A l'insertion la valeur est égale à 1 (1 = session ouverte).
- La procédure « P4 » : cette procédure permet de vérifier la validité de la session. Le paramètre d'entrée est l'ID de session. La première vérification consiste à contrôler si la date et l'heure actuelle sont inférieures à la date et l'heure de l'expiration de session (attribut « DateExpirationSession » de la table « T2 »). Si la date et l'heure actuelle sont supérieures à la date d'expiration de session alors la procédure retourne la valeur « -30 » qui signifie que la session a expiré. Dans le cas où la date et l'heure actuelle sont inférieures à la date d'expiration de session alors il faut passer à la seconde vérification. La seconde vérification consiste à contrôler le quota de requête de la session. Pour ce faire, il faut contrôler que le nombre d'accès (attribut « NombreAcces » de la table « T2 ») soit inférieur ou égal au quota défini dans la table « T1 ». Si le quota de la session est supérieur au quota défini alors la procédure retourne la valeur « -31 » qui signifie que l'utilisateur a atteint le nombre maximum de requêtes sinon elle retourne la valeur « 0 » indiquant que la session est valide. Si la procédure retourne la valeur « -30 » ou « -31 » alors il faut faire appel à la procédure

stockée « P5 ». Si la session est valide (valeur de retour égal à 0), il faut passer à la procédure stockée « P6 ».

- La procédure « P5 » : cette procédure permet de fermer une session (logout). Le paramètre d'entrée est l'ID de session. Pour fermer une session, il faut mettre à jour les informations relatives à celle-ci. L'attribut « DateFinSession » est mis à jour à la date et l'heure actuelle et l'attribut « Statut » passe à 0 (0 = session fermée).
- La procédure « P6 » : cette procédure permet de contrôler que l'utilisateur qui fait la requête correspond à l'adhérent qui a ouvert la session ou qu'il fait partie du même groupe que ce dernier. Cette procédure de vérification permet de résoudre le problème de sécurité expliqué au début du document, en effet, ce contrôle garantie que l'adhérent « A » puisse accéder uniquement à ces informations et non aux informations de l'adhérant « B » (à moins que l'adhérent « A » connaisse les identifiants de l'adhérant « B » et ses informations de session). Le paramètre d'entrée de cette procédure de vérification est l'ID de session. La procédure retourne la valeur « -2 » si l'adhérent n'est pas celui qui a ouvert la session. Si l'adhérent qui fait la demande est celui qui a ouvert la session alors la procédure retourne la valeur « 0 ». Dans ce cas, l'adhérent peut faire appel aux lectures de données.
- La procédure « P7 » : cette procédure permet de mettre à jour la session. Le paramètre d'entrée est l'ID de session. Dès lors que l'adhérent demande une lecture de données et qu'il passe toutes les procédures de vérifications, la session est mise à jour. Le nombre d'accès (attribut « NombreAcces » de la table « T2 ») est incrémenté et la date de fin d'expiration (attribut « DateExpirationSession » de la table « T2 ») est mise à jour, car celle-ci est étendue.
- La procédure « P8 » : cette procédure permet de lire les données que l'adhérent demande. L'adhérent peut lire plusieurs informations et pour chaque information une procédure stockée lui est dédiée :
    - la procédure « P8.1 » permet de lire les données relatives au groupe d'expédition d'un adhérent.
    - la procédure « P8.2 » permet de lire les données relatives aux informations administratives d'un adhérent.
    - la procédure « P8.3 » permet de lire les données relatives aux programmes d'un adhérent
    - la procédure « P8.4 » permet de lire les données relatives aux paramètres d'un programme d'un adhérent.
- La procédure « P9 » : cette procédure permet de tracer les demandes des adhérents aux services Web. Les demandes sont enregistrées dans la table « T3 ». Un traçage (log) est caractérisé comme ceci :
    - ID : identifiant du log (incrémentation automatique)
    - IDSession : identifiant de la session, valeur fournie en paramètre
    - ServiceInvoquer : nom de la procédure invoquée (« P5 » par exemple), cette valeur est fournie en paramètre
    - ServiceParametres : paramètre associé au service invoqué (exemple : erreur et détail). Cette valeur est fournie en paramètre.
    - IDAdherent : identifiant de l'adhérent, valeur fournie en paramètre

- Date : date et heure de la demande

Les informations conservées dans le « log » sont un moyen supplémentaire d'augmenter la sécurité. Il est ainsi possible de détecter des tiers qui essayeraient d'utiliser des services sans les autorisations adéquates. De plus, il permet de faire des statistiques et d'analyser les informations les plus pertinentes pour les adhérents. C'est un moyen pour le CSCQ d'améliorer les services qu'il propose.

4.4 Service Web

Cette section a pour but de présenter le service Web développé tout au long de ce projet. Ce dernier a été développé en suivant une architecture REST à l'aide des spécifications JAX-RS. Ces spécifications permettent de créer des services Web REST en Java. Pour implémenter ces spécifications, la bibliothèque « Jersey » a été utilisée. D'autres bibliothèques permettent d'implémenter JAX-RS notamment « RESTEasy » ou « Resltet ». Cependant, « Jersey » sort du lot par son efficacité et sa forte utilisation, et de plus, cette bibliothèque est maintenue et développée par Oracle. Du côté serveur, cette bibliothèque fournit une servlet qui parcourt automatiquement les classes pour identifier les ressources. Du côté client, cette bibliothèque fournit les méthodes nécessaires pour communiquer avec le service.

Pour résumer, JAX-RS se repose sur un système d'annotation pour gérer les services Web de type REST et la bibliothèque « Jersey » permet d'assurer son implémentation en scannant les classes annotées pour identifier les ressources. Le tableau ci-dessous liste les annotations les plus utilisées.

Tableau 14 : Annotations JAX-RS

Annotation	Description
@GET	Indique qu'il s'agit d'une méthode GET
@Produces	Définit le type de représentation (XML, JSON, …)
@Path	Définit le chemin à partir de l'URL de base
@PathParam	Permet de récupérer les paramètres dans l'URL. Paramètres liés au chemin.
@QueryParam	Permet de récupérer les paramètres (clé/valeur) dans l'URL. Paramètres liés à la requête.

@POST	Indique qu'il s'agit d'une méthode POST
@Consumes	Définit le type de représentation accepté par la méthode
@PUT	Indique qu'il s'agit d'une méthode PUT
@DELETE	Indique qu'il s'agit d'une méthode DELETE

4.4.1 Développement de l'application

Cette application a été développée avec l'environnement de développement « Eclipse IDE for Java EE Developers ». La version 7 de Java a été utilisée pour le développement et la compilation de cette application.

Dans « Eclispe », un projet « Dynamic Web Projet » a été créé et nommé « cscqinfo ». La première étape après la création du projet a été d'ajouter plusieurs bibliothèques à ce dernier :

- la bibliothèque « Jersey ». Comme expliqué précédemment cette bibliothèque permet d'implémenter les spécifications JAX-RS.
- la bibliothèque « ASM » qui permet de manipuler et analyser du « bytecode » Java. En d'autres termes, cette bibliothèque est utilisée pour modifier ou créer des classes Java sous formes binaires.
- la bibliothèque « JDBC for SQL Server ». Cette bibliothèque fournit une connectivité aux bases de données SQL Serveur.
- la bibliothèque « LibWS ». Cette dernière a été développée en interne et permet d'exécuter des procédures stockées dans la base de données.

Les applications JAX-RS doivent inclure un fichier qui se nomme « web.xml ». En effet, ce fichier permet d'associer un chemin (URL) à une servlet. Ce fichier doit être situé à la racine du dossier « WEBINF ». La figure suivante correspond au fichier « web.xml » de l'application.

Figure 49 : Fichier web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">
  <display-name>RESTWebApp</display-name>
  <servlet>
    <servlet-name>jersey-servlet</servlet-name>
    <servlet-class>
            com.sun.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>jersey-servlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Pour mieux organiser le projet, celui-ci a été structuré en quatre packages :

- « bdd » contient les éléments qui vont faire appel à la base de données
- « domaine » contient les entités décrivant les objets de l'application
- « métier » contient les éléments de calculs propres et uniques à l'application
- « service » contient les classes annotées selon les spécifications JAX-RS

La section suivante détaille le package « service ». Ce dernier est l'élément cœur de l'application, car les classes qu'il contient permettent de fournir les ressources à l'adhérent.

4.4.2 Services

4.4.2.1 Ouverture d'une session (login)

Ce service permet à un adhérent de s'authentifier afin de créer une session.

Figure 50 : Service « ouvrir une session »

```
@GET
@Path("login")
@Produces(MediaType.APPLICATION_JSON)
public Session login(@HeaderParam("authorization") String authentification)
throws Exception  {
    return DaoLoginLogout.getSession(authentification);
}
```

La figure ci-dessous permet de distinguer trois annotations :

- @GET qui indique qu'il s'agit d'une lecture. En effet, en appelant ce service l'adhérent récupère une session.
- @Path qui permet de construire le chemin vers la ressource. Dans ce cadre, cette ressource est accessible via l'URL suivante : {serveur}/{nomApplication}/{login}.
- @Produces qui indique la représentation possible de la ressource. Il a été décidé de fournir une représentation JSON uniquement. Mais une ressource peut avoir plusieurs représentations : @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML)}).

Cette fonction reçoit en paramètre d'entrée un « @HeaderParam » qui permet de mettre en place l'authentification basique HTTP. Comme expliqué dans la section sécurité de ce document, cette authentification est uniquement efficace avec le protocole de communication HTTPS afin de garantir des échanges cryptés entre le client et le serveur. La fonction « getSession » permet de récupérer une session, mais celle-ci possède quelques vérifications préliminaires. En effet, cette fonction va faire appel dans un premier temps à la procédure stockée « P1 » qui permet de vérifier la disponibilité du service Web. Si l'application n'est pas disponible alors la fonction « getSession » retourne l'exception suivante :

throw new WebApplicationException(Statut.SERVICE_UNAVAILABLE)

Cette exception retourne le code d'erreur HTTP 503 qui signifie que le service est temporairement indisponible ou en maintenance.

Dans un deuxième temps, si le service est disponible, la fonction « getSession » va contrôler que l'adhérent a bien fourni une authentification valide grâce à la procédure stockée « P2 ». Dans le cas où l'adhérent ne fournit pas d'authentification ou s'il fournit une authentification erronée, la fonction « getSession » retourne l'exception suivante :

throw new WebApplicationException(Statut.UNAUTHORIZED)

Cette exception retourne le code d'erreur HTTP 401 qui signifie qu'une authentification valide est nécessaire pour accéder à la ressource.

Dans un troisième temps, après avoir passé les deux vérifications expliquées ci-dessous la fonction « getSession » retourne une représentation JSON de la ressource session via la procédure stockée « P3 » qui permet d'ouvrir une session. Lorsque qu'une requête est exécutée avec succès le code HTTP 200 est retourné.

4.4.2.2 Fermeture d'une session (logout)

Ce service permet à un adhérent de se déconnecter et de fermer une session.

Figure 51 : Service « fermeture d'une session »

```
@GET
@Path("logout")
public void logout(@HeaderParam("authorization")
String authentificationHeader,
@QueryParam("session") int session)
throws Exception  {
    DaoLoginLogout.fermerSession(session, authentificationHeader);
}
```

Dans ce cadre ce service est disponible à l'aide de l'URL suivante : {serveur}/{nomApplication}/{logout}. L'adhérent doit fournir en paramètre le numéro de session. La méthode « fermerSession » contrôle la disponibilité de l'application et l'authentification de l'adhérent avant de fermer une session. Ces deux contrôles sont les mêmes que pour le service « login ». Lorsque la session a été fermée, la méthode « fermerSession » renvoie le code HTTP 204 qui indique que la requête a été traitée avec succès, mais qu'il n'y a pas d'information à renvoyer.

4.4.2.3 Lecture du groupe d'expédition

Ce service permet à un adhérent de récupérer la liste des adhérents qui font partie du même groupe d'expédition.

Figure 52 : Service « lecture du groupe d'expédition »

```
@GET
@Path("/adherents/{adherentConnexion}/expeditions")
@Produces(MediaType.APPLICATION_JSON )
public AdherentListe getAdherents(
    @QueryParam("session") int session,
    @PathParam("adherentConnexion") int adherentConnexion,
    @HeaderParam("authorization") String authentificationHeader)
    throws Exception{
        return DaoAdherent.getAdherents(adherentConnexion,
            session,authentificationHeader);
}
```

Pour obtenir cette ressource un adhérent doit faire appel à l'URL suivante : {serveur}/{nomApplication}/{adherents}/{adherentConnexion}/{expeditions}. La représentation de la ressource est en JSON. La fonction « getAdherents » permet de récupérer une collection d'adhérents, mais des contrôles de validité sont réalisés au préalable.

Dans un premier temps, la fonction vérifie la disponibilité de l'application, dans un second temps elle vérifie l'authentification de l'adhérent et dans un troisième temps elle vérifie la validité de la session. La validité de la session est contrôlée à l'aide de la procédure stockée « P4 ». Si la procédure « P4 » indique que la session a expiré alors la fonction « getAdherents » renvoie l'exception suivante :

throw new WebApplicationException(440)

Cette exception retourne le code HTTP 440 qui indique à l'utilisateur que la session a expiré et qu'il doit se connecter à nouveau. Si la procédure stockée indique que le quota de la session a été dépassé alors la fonction « getAdherents » retourne l'exception :

throw new WebApplicationException(429)

Cette exception retourne le code HTTP 429 qui indique que le client a effectué trop de requêtes au long de sa session. Si la procédure « P4 » ne relève aucune exception alors il est nécessaire de contrôler que l'adhérent qui fait la demande est bien celui qui a ouvert la session. Si l'adhérent qui fait la demande n'est pas celui qui a ouvert la session alors la fonction « getAdherents » retourne l'exception :

throw new WebApplicationException(Statut.UNAUTHORIZED)

Si toutes les vérifications sont passées avec succès alors la fonction « getAdherents » appelle la procédure stockée « P8.1 » qui permet de récupérer la liste des adhérents du groupe d'expédition.

4.4.2.4 Lecture des informations administratives

Ce service permet à un adhérent de consulter ses informations administratives.

Figure 53 : Service « lecture informations administratives »

```
@GET
@Path("/adherents/{adherentConnexion}/expeditions/
    {adherent}/informations")
@Produces(MediaType.APPLICATION_JSON )
public AdherentComplet getAdherent(
    @QueryParam("session") int session,
        @DefaultValue("-1") @QueryParam("langue") int langue,
        @PathParam("adherentConnexion") int numeroAdherentConnexion,
        @PathParam("adherent") int numeroAdherent,
        @HeaderParam("authorization") String authentificationHeader)
        throws Exception{
            return DaoAdherent.getAdherent(numeroAdherent, session,
        langue, authentificationHeader);
}
```

Pour obtenir cette ressource un adhérent doit faire appel à l'URL suivante : {serveur}/{nomApplication}/{adherents}/{adherentConnexion}/{expeditions}/{adherent}/{informations}. Ce service possède un paramètre de plus qui est la langue. En effet, un adhérent peut obtenir son profil en quatre langues soit en français, anglais, italien ou allemand. La valeur « -1 » signifie qu'on prend par défaut la langue de l'adhérent. Tout comme pour les services précédents, ce service possède des contrôles préalables avant de fournir la ressource (informations administratives de l'adhérent). En effet, la fonction « getAdherent » vérifie la disponibilité de l'application, l'authentification de l'adhérent, la validité de la session ainsi qu'une cohérence de demande ce qui permet de vérifier que l'adhérent qui demande les informations est celui qui a ouvert la session. Si toutes ces conditions sont respectées alors la fonction « getAdherent » fait appel à la procédure stockée « P8.2 » qui permet de récupérer les informations administratives de l'adhérent.

4.4.2.5 Lecture des programmes

Ce service permet à un adhérent de consulter les programmes auxquels il est inscrit au CSCQ.

Figure 54 : Service « lecture programmes »

```
@GET
@Path("/adherents/{adherentConnexion}/expeditions/
    {adherent}/programmes")
@Produces(MediaType.APPLICATION_JSON)
public ProgrammeListe getProgrammes(
    @QueryParam("session") int session,
    @DefaultValue("-1") @QueryParam("langue") int langue,
    @PathParam("adherentConnexion") int adherentConnexion,
    @PathParam("adherent") int adherent,
    @HeaderParam("authorization") String authentificationHeader)
    throws Exception{
        return DaoProgramme.getProgrammes(adherent, session,
            langue, authentificationHeader);
}
```

Pour obtenir cette ressource un adhérent doit faire appel à l'URL suivante : {serveur}/{nomApplication}/{adherents}/{adherentConnexion}/{expeditions}/{adherent}/{programmes}. Ce service possède la même structure que le service précédent (lecture des informations administratives d'un adhérent). En effet, les conditions de vérifications sont les mêmes, l'unique différence se trouve au niveau de la ressource. Ce service renvoie une collection de programmes.

4.4.2.6 Lecture des paramètres d'un programme

Ce service permet à un adhérent de consulter les paramètres associés à un programme.

Figure 55 : Service « lecture paramètres »

```
@GET
@Path("/adherents/{adherentConnexion}/expeditions/{adherent}/
    programmes/{codeProgramme}/parametres")
@Produces(MediaType.APPLICATION_JSON)
public ParametreListe getParametres(
    @QueryParam("session") int session,
    @DefaultValue("-1") @QueryParam("langue") int langue,
    @PathParam("adherentConnexion") int numeroAdherentConnexion,
    @PathParam("adherent") int adherent,
    @PathParam("codeProgramme") String codeProgramme,
    @HeaderParam("authorization") String authString)
    throws Exception{
        return DaoParametre.getParametres(adherent, session,
            codeProgramme, langue, authString);
}
```

Ce service possède un paramètre supplémentaire qui est le code du programme qui permet de récupère les paramètres de ce dernier. Ce service possède également les contrôles de validité vus précédemment. Dans ce cadre, après avoir passé tous les

contrôles de validité la fonction « getParametres » retourne la collection de paramètres du programme fourni en paramètre.

4.5 Environnement de test

Il est important de noter qu'un simple navigateur peut jouer le rôle de client pour un service REST. Cependant, un navigateur basique ne propose pas forcément toutes les fonctionnalités avancées pour tester un tel service. En effet, lorsque qu'il est nécessaire de tester l'authentification HTTP ou des méthodes « POST », « PUT » et « DELETE », un navigateur devient inopérant. Dans ce cadre, les tests du service Web développé au long de ce travail ont été réalisés à l'aide du logiciel « Postman ». Ce logiciel est une extension (plugin) du navigateur « Google Chrome ». Ce dernier propose une interface graphique agréable et intuitive. Les fonctionnalités de ce logiciel couvrent l'ensemble des caractéristiques qu'un service REST peut proposer.

4.6 Application Android

4.6.1 Avant-propos

L'objectif final de ce développement est d'offrir une application mobile permettant d'interagir avec le service Web. En effet, l'utilisation d'un navigateur ou d'un logiciel tiers pour consulter un service Web n'est pas très agréable pour un utilisateur final (adhérent du CSCQ). Dans ce cadre, une application Android a été réalisée pendant ce travail de Bachelor et une application IOS et une application Web seront développées par la suite.

L'application Android a été développée via l'IDE « Android Studio ». Cet environnement de développement est maintenu et conseillé par Google depuis 2015. Auparavant, Google conseillait aux développeurs Android d'utiliser l'IDE « Eclipse ». Afin de pouvoir développer des applications Android sur « Eclipse », il est nécessaire d'ajouter le plugin « ADT ». Il est important de noter que Google a arrêté le support de ce plugin à la fin de l'année 2015, ce qui force les développeurs à migrer sur Android Studio.

Android est un système d'exploitation mobile basée sur le noyau Linux qui permet de construire des applications dans un environnement Java. Avant de développer une application, il est important de cibler la version. En effet, plusieurs versions Android ont déjà vu le jour et à chaque nouvelle version, Google propose encore plus de fonctionnalités. Dans ce cadre, il est nécessaire de définir une version minimum d'Android pour que les fonctionnalités offertes par l'application puissent être exécutées sur un nombre élargi d'appareils. Le tableau ci-dessous détaille le pourcentage d'appareil Android pour chaque version.

Tableau 15 : Versions Android

Version	Nom de code	Date de sortie	Version API	%
2.2-2.2.3	Froyo	20 mai 2010	8	0,1 %
2.3.3 - 2.3.7	Gingerbread	6 décembre 2010	10	1,7 %
4.0.3 - 4.0.4	Ice Cream Sandwich	19 octobre 2011	15	1,6 %
4.1.x	Jelly Bean	9 juillet 2012	16	6 %
4.2.x		13 novembre 2012	17	8,3 %
4.3.x		24 juillet 2013	18	2,4 %
4.4-4.4.4	KitKat	31 octobre 2013	19	29,2 %
5.0-5.0.2	Lollipop	3 novembre 2014	21	14,1 %
5.1.x		9 mars 2015	22	21,4 %
6.0	Marshmallow	8 octobre 2015	23	15,2 %

Source : WIKIPEDIA, 29 août 2016

L'application Android du CSCQ est développée avec l'API 18, ce qui permet de cibler les appareils avec une version 4.3.X et supérieurs soit environ 80% des utilisateurs.

4.6.2 Développement de l'application

Après mon travail de Bachelor, l'application Android va être maintenu par le CSCQ, il est donc important de séparer le projet en quatre packages :

- « bdd » qui contient les éléments faisant appel au service Web
- « domaine » qui contient les entités décrivant les objets de l'application
- « metier » qui contient les éléments de calculs propres et uniques à l'application
- « presentation » qui contient les activités (interface graphique) pour gérer l'interaction entre l'utilisateur et l'application

En découpant le projet de cette manière, le développeur futur de l'application pourra cibler une erreur ou effectuer une mise à jour rapidement.

L'application Android est disponible en quatre langues et l'adhérent peut changer la langue via le menu de l'application dynamiquement à tout moment. Les langues proposées sont les suivantes :

- Français
- Anglais
- Italien
- Allemand

Les captures d'écran sont présentées en français. Elles sont issues d'une base de données de test qui inclut des données fictives. Seul l'exemple de l'activité authentification est présenté dans les quatre langues.

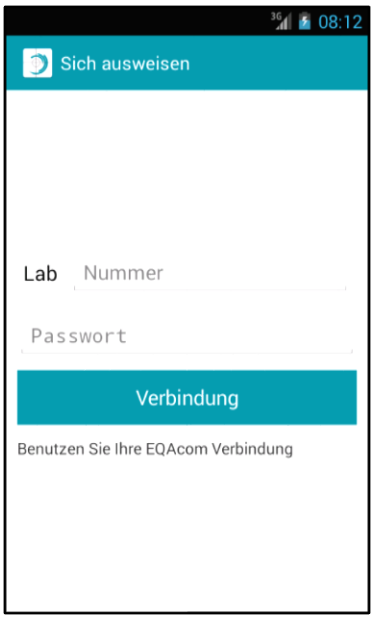
Les icônes utilisées dans l'application ont été téléchargées sur le site https://icons8.com et sont libres pour un usage personnel et commercial. L'unique contrainte est de fournir un lien vers le site https://icons8.com/ depuis une activité de l'application mobile.

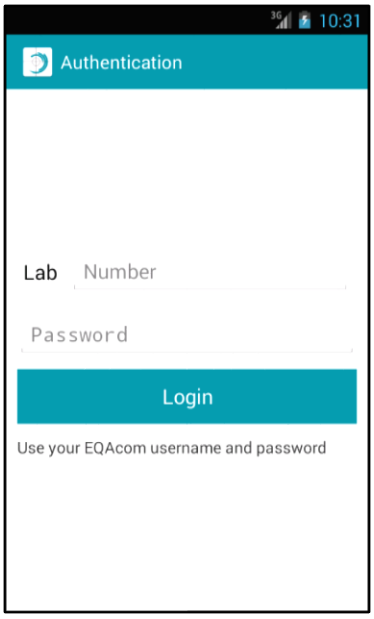
4.6.2.1 Activité authentification (login)

Les figures ci-dessous montrent l'activité obtenue après le lancement de l'application selon les quatre langues disponibles. Cette activité permet à un adhérent du CSCQ de s'authentifier afin de créer une session. Pour ce faire, il doit fournir ses informations d'authentification c'est-à-dire son numéro d'adhérent et son mot de passe. Pour valider les informations saisies par l'adhérent, l'application fait appel au service Web développé et décrit précédemment. Cet appel retourne l'un des trois codes suivants :

- 200 qui est accompagné d'un identificateur de session au format JSON
- 503 ou 401 qui retournent une exception et provoquent l'affichage d'un message d'erreur par l'application

Figure 56 : Activités authentification



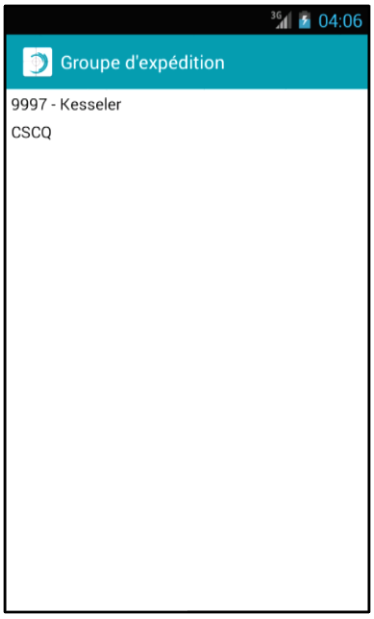


4.6.2.2 Activité groupe d'expédition

Cette activité est accessible après l'authentification d'un adhérent. Elle permet d'afficher dans une liste les adhérents qui font partie du même groupe d'expédition que l'adhérent authentifié. L'appel au service Web retourne l'un des quatre codes suivants :

- 200 qui est accompagné par une collection d'adhérents au format JSON. Cette collection est ensuite traitée et affichée dans une liste
- 503, 440 et 429 qui sont des exceptions et sont traitées en tant que message d'erreur par l'application

Figure 57 : Activité groupe d'expédition



4.6.2.3 Activité menu

A partir de la liste des adhérents du groupe d'expédition, l'utilisateur peut en sélectionner un pour consulter son profil. Le menu de l'application présenté ci-dessous est alors affiché. Il inclut des boutons permettant de :

- consulter les informations administratives
- lister les programmes auxquels l'adhérent choisi est inscrit
- obtenir les informations de contact sur CSCQ
- fermer la session

Cette activité n'interagit pas avec le service Web, mais permet à l'adhérent de naviguer dans l'application.

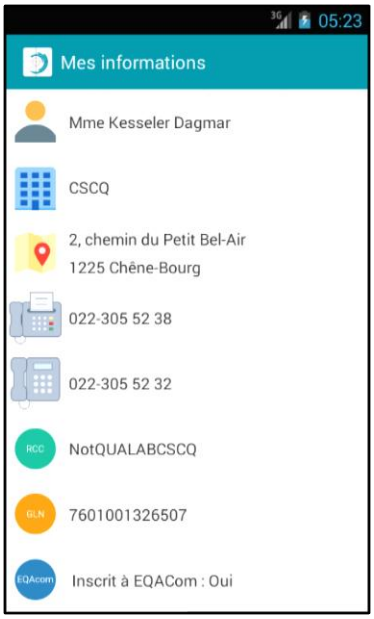
Figure 58 : Activité menu

4.6.2.4 Activité informations administratives

Cette activité permet à l'adhérent de consulter ses informations administratives. L'appel au service Web retourne l'un des quatre codes suivants :

- 200 qui est accompagné des informations administratives de l'adhérent au format JSON. Cette représentation est traitée et affichée dans l'activité
- 503, 440 et 429 qui sont des exceptions et sont traitées en tant que message d'erreur par l'application

Figure 59 : Activité informations administratives



4.6.2.5 Activité programmes

Cette activité permet d'afficher les programmes auxquels l'adhérent s'est inscrit. L'appel au service Web retourne l'un des quatre codes suivants :

- 200 qui est accompagné par une collection de programmes au format JSON. Cette collection est ensuite traitée et affichée dans une liste
- 503, 440 et 429 qui sont des exceptions et sont traitées en tant que message d'erreur par l'application

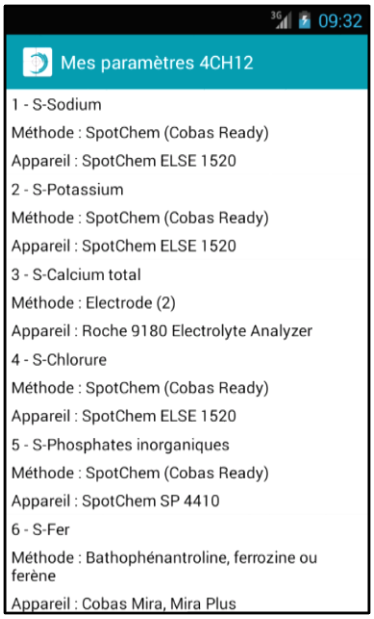
Figure 60 : Activité programmes

4.6.2.6 Activité paramètres

A partir de la liste des programmes, l'utilisateur peut en sélectionner un pour consulter les paramètres auxquels il s'est inscrit. Cette activité permet d'afficher la collection de paramètres dans une liste. L'appel au service Web retourne l'un des quatre codes suivants :

- 200 qui est accompagné par une collection de paramètres au format JSON. Cette collection est ensuite traitée et affichée dans une liste
- 503, 440 et 429 qui sont des exceptions et sont traitées en tant que message d'erreur par l'application

Figure 61 : Activité paramètres



4.6.2.7 Activité « Contactez le CSCQ »

L'activation du bouton « Contactez le CSCQ » permet d'afficher les informations de contact du CSCQ dans une nouvelle activité.

La sélection de l'adresse affiche dans une application tierce telle Google Map la localisation du CSCQ. Si l'application est utilisée sur un téléphone mobile, la sélection du numéro de téléphone affiche la fenêtre d'appel avec le numéro du CSCQ dans la langue sélectionnée. La sélection de l'adresse ouvre le client mail par défaut et permet à l'utilisateur de rédiger et envoyer un mail au CSCQ. La sélection du site Internet ouvre le navigateur par défaut et affiche la page d'accueil du site du CSCQ.

Figure 62 : Activité « Contactez le CSCQ »

4.6.2.8 Traitement des messages d'erreurs

4.6.2.8.1 Erreur 503

Lorsque le service Web retourne l'exception 503, l'application affiche une boîte de dialogue indiquant à l'adhérent que le service est temporairement indisponible ou en maintenance.

Figure 63 : Erreur 503

4.6.2.8.2 Erreur 401

L'erreur 401 est retournée lorsque l'adhérent saisit une authentification erronée (nom d'utilisateur et mot de passe).

Figure 64 : Erreur 401

4.6.2.8.3 Erreur 440

L'erreur 440 est retournée lorsque la session de l'adhérent a expiré, suite à une inactivité de plus de 10 minutes. Ce délai peut être ajusté par l'administrateur du service Web.

Figure 65 : Erreur 440

4.6.2.8.4 Erreur 429

L'erreur 429 est retournée lorsque le nombre de requêtes pour une session demandée par l'adhérent a dépassé un quota prédéfini. Ce quota peut être ajusté par l'administrateur du service Web.

Figure 66 : Erreur 429

5. Conclusion

Depuis le début des années 1990, Internet et surtout le Web ont révolutionné les échanges tant au niveau des individus dans la sphère privée qu'au niveau professionnel. Les modes de communication ont complètement changé. Le transport de l'information se fait, de nos jours, principalement sous forme électronique à la vitesse de la « lumière ». Les informations sont de plus disponibles quasiment en tout lieu et à toute heure.

Les entreprises doivent donc s'adapter à ces changements et proposer de nouveaux services et moyens de communication avec leurs clients. Une des solutions maintenant fort répandue est la mise en place d'applications mobiles s'appuyant sur des services Web. En effet, cette technologie a l'avantage d'offrir une nouvelle impulsion concernant l'interopérabilité des systèmes d'information et permet aux entreprises d'ouvrir facilement une partie de leurs processus métier vers leurs clients et partenaires. Les services Web permettent aux développeurs de se concentrer davantage sur le développement des processus métier plutôt que sur les problèmes techniques liés à l'intégration et l'interopérabilité. Les précurseurs tels que CORBA, DCOM ou RMI ont du mal à rivaliser avec les services Web, car ils sont peu adaptés aux besoins d'aujourd'hui. Un frein à leur diffusion est le couplage fort entre les objets traités rendant la mise en place et la maintenance des applications très complexes et nécessitant la plupart du temps une formation coûteuse des développeurs.

Une des questions à se poser lors de la mise en place d'un service Web est le choix de l'architecture. SOAP et REST sont les deux technologies les plus utilisées actuellement, ayant chacune leurs avantages et inconvénients. REST est plus facile à mettre en place et propose une meilleure interopérabilité tandis que SOAP est un peu plus complexe, mais propose des solutions de sécurité intégrées. Techniquement ces deux solutions se valent, mais REST est plus adaptée pour les terminaux mobiles (smartphones, tablettes, …) et les applications Web principalement grâce à la compacité des données échangées. Le service Web du CSCQ a donc été construit avec un style architectural REST avec un format d'échange en JSON.

Une des contraintes de ce projet est de garantir la confidentialité des informations accédées par le service Web. Ce dernier doit être uniquement accessible aux utilisateurs autorisés, chacun d'eux ne pouvant bien évidement consulter que ses propres informations. Cette restriction a été traitée un niveau du réseau et au niveau applicatif. Au niveau réseau, il a été décidé d'utiliser le protocole HTTPS pour la communication entre le serveur et le client. Cette solution garantit la confidentialité des messages

échangés entre les deux partenaires. Au niveau applicatif, chaque demande du client vers le serveur requiert une authentification garantissant que le service Web est uniquement accessible aux utilisateurs ayant les droits adéquats. Cette authentification est gérée via l'authentification basique HTTP. Elle n'est efficace qu'avec le protocole de communication HTTPS. Pour renforcer la sécurité niveau applicatif, une gestion de session a été développée pour compléter l'architecture REST. Cette gestion inclut une limitation du temps d'ouverture d'une session et du nombre de requêtes faites par l'utilisateur.

Côté client une application mobile sous Android a été développée. Elle interagit avec le service Web mis en place et offre aux adhérents du CSCQ la possibilité de consulter leur profil (informations administratives, expédition, programmes et paramètres) dans un environnement sécurisé.

Par la suite, l'application mobile sera également adaptée pour les terminaux IOS (Apple). Une application Web accessible par un simple navigateur sera aussi proposée. L'architecture mise en place est actuellement orientée vers des applications « adhérents », mais elle va aussi servir aux échanges d'informations entre le CSCQ et ses partenaires notamment dans le cadre d'organisation d'enquêtes en collaboration ou en sous-traitance.

6. Expérience personnelle

Dans le cadre de mes études à la HEG, j'ai eu l'occasion de suivre pendant quelques semaines un cours dédié au fonctionnement des services Web de type REST. De plus, j'ai eu l'occasion de travailler sur plusieurs projets faisant appel à des services Web sans pouvoir analyser comment les mécanismes étaient gérés côté serveur. De fait, lorsque ce travail m'a été confié, j'étais très enthousiaste à l'idée d'approfondir mes connaissances dans ce domaine, d'autant plus que cette technologie est aujourd'hui très demandée due à la forte augmentation de l'utilisation des périphériques mobiles.

Tout au long de mes recherches, j'ai pu facilement trouver des informations sur ces nouvelles technologies et pu ainsi résoudre les problèmes rencontrés dans la mise en place de l'architecture (REST). Cependant, les informations sur les précurseurs tels CORBA, DCOM et RMI ont été plus difficiles à trouver.

Au cours du développement, j'ai pu constater que la construction d'un service Web se fait très rapidement, mais lorsqu'on intègre les contraintes de sécurité, le développement et la mise en place deviennent tout de suite plus complexes. D'autant plus que la sécurité est un « métier » à part entière dans le monde de l'informatique.

Enfin, j'ai eu la chance de mettre en pratique mes connaissances dans un projet concret pour le CSCQ et permettre à l'entreprise d'offrir un service supplémentaire à ses adhérents. Cela sera une fierté pour moi de savoir que le service est utilisé et maintenu au cours de ces prochaines années.

Bibliographie

CSCQ, 2015. Manuel. [Consulté le 27.06.2016]. Disponible à l'adresse : http://www.cscq.ch/SiteCSCQ/FichierPDF_FR/M-Manuel.pdf

ZDNET, 2016. Chiffres clé : le marché des tablettes. [Consulté le 28.06.2016]. Disponible à l'adresse : http://www.zdnet.fr/actualites/chiffres-cles-le-marche-des-tablettes-39789571.htm

REFES, Chabane, 2016. Les services Web. [Consulté le 28.06.2016]. Disponible à l'adresse : https://openclassrooms.com/courses/les-services-web

WIKIPEDIA, 2004. Service Web. [Consulté le 28.06.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Service_web

STATTNER, Erick, 2015. Applications et Services Web : Architecture REST. [Consulté le 28.06.2016]. Disponible à l'adresse : http://erickstattner.com/enseignement/2013-14/M2_AppliEtServicesWeb_2013-14.pdf

EDOUARD, Amosse, 2014.Comprendre l'architecture des Web Services REST. [Consulté le 28.06.2016]. Disponible à l'adresse : http://miage.unice.fr/@api/deki/files/2369/=webservice_REST.pdf

WIKIPEDIA, 2004. Common Object Request Broker Architecture. [Consulté le 16.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Service_web

THE OPEN GROUP, 1997. OMG Mapping. [Consulté le 16.08.2016]. Disponible à l'adresse : http://pubs.opengroup.org/onlinepubs/9279299/apdxa.htm

LALOUM, Yves, 2003. Présentation de l'architecture CROBA. [Consulté le 16.08.2016]. Disponible à l'adresse : http://deptinfo.cnam.fr/Enseignement/CycleSpecialisation/ISA/ISCS/Laloum/CORBA-cnam.pdf

SEONGKI, Kim, SANGYOUNG, Han, 2006. Performance comparison of DCOM, CORBA and Web service. [Consulté le 16.08.2016]. Disponible à l'adresse : http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2405&rep=rep1&type=pdf

WIKIPEDIA, 2008. Distributed Component Object Model. [Consulté le 15.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Distributed_Component_Object_Model

DURAND-POUDRET, Yoann, 2015. CORBA vs COM. [Consulté le 15.08.2016]. Disponible à l'adresse : http://docplayer.fr/16200989-Corba-vs-com-urbanisation-des-si-nfe107-fiche-de-lecture-y-durand-poudret.html

WIKIPEDIA, 2004. Remote method invocation. [Consulté le 15.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Remote_method_invocation

JAVATPOINT, 2014. RMI (Remote Method Invocation). [Consulté le 15.08.2016]. Disponible à l'adresse : http://www.javatpoint.com/RMI

THAKUR, Aniket, 2015. Understanding Java Remote Method Invocation (RMI). [Consulté le 15.08.2016]. Disponible à l'adresse : http://opensourceforgeeks.blogspot.ch/2015/02/understanding-java-remote-method.html

WIKIPEDIA, 2006. Representational State Transfer. [Consulté le 15.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Representational_state_transfer

CHUAN, Chua Hock, 2009. HTTP (HyperText Transfer Protocol). [Consulté le 30.06.2016]. Disponible à l'adresse : https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

JOSHI, Prateek, 2014. URL vs URI vs URN. [Consulté le 30.06.2016]. Disponible à l'adresse : https://prateekvjoshi.com/2014/02/22/url-vs-uri-vs-urn/

EVANS, Nathan, 2011. WebSockets versus REST… fight!. [Consulté le 30.06.2016]. Disponible à l'adresse : https://nbevans.wordpress.com/2011/12/16/websockets-versus-rest-fight/

ROBERT, Mikael, 2011. Initiation à la sécurité des Web Services [Consulté le 11.08.2016]. Disponible à l'adresse : http://blog.octo.com/initiation-a-la-securite-des-web-services/

WIKIPEDIA, 2004. Base64. [Consulté le 11.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Base64

OPENCLASSROMS, 2004. Tout en Base64. [Consulté le 11.08.2016]. Disponible à l'adresse : https://openclassrooms.com/forum/sujet/defis-8-tout-en-base64-19054?page=1#message-6910110

WIKIPEDIA, 2008. OAuth. [Consulté le 11.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/OAuth

RATSITOBAINA, Hoby, 2014. OAuth : Comment ça marche ?.. [Consulté le 11.08.2016]. Disponible à l'adresse : http://blog.netapsys.fr/oauth-comment-ca-marche/comment-page-1/

REINKE, Johann, 2016. Understanding OAuth2. [Consulté le 11.08.2016]. Disponible à l'adresse : http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/

WIKIPEDIA, 2007. WS-Security. [Consulté le 12.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/WS-Security

SEELY, Scott, 2002. Understanding WS-Security. [Consulté le 12.08.2016]. Disponible à l'adresse : https://msdn.microsoft.com/en-us/library/ms977327.aspx

CHASE, Nicholas, 2006. Understanding web services specifications, Part 4: WS-Security. [Consulté le 12.08.2016]. Disponible à l'adresse : http://www.ibm.com/developerworks/webservices/tutorials/ws-understand-web-services4/ws-understand-web-services4.html

OASIS, 2006. Web Services Security:SOAP Message Security 1.1 (WS-Security 2004). [Consulté le 12.08.2016]. Disponible à l'adresse : https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

WIKIPEDIA, 2002. Extensible Markup Language. [Consulté le 04.07.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Extensible_Markup_Language

SAUVAGE, Sébastien, 2011. C'est quoi Xml. [Consulté le 04.07.2016]. Disponible à l'adresse : http://sebsauvage.net/comprendre/xml/

WIKIPEDIA, 2006. JavaScript Object Notation. [Consulté le 04.07.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

JSON ORG. Présentation de JSON. [Consulté le 04.07.2016]. Disponible à l'adresse : http://www.json.org/json-fr.html

ASCIITABLE. ASCII Table and Description. [Consulté le 11.08.2016]. Disponible à l'adresse : http://www.asciitable.com/

WIKIPEDIA, 2007. Proxy inverse. [Consulté le 18.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Proxy_inverse

WIKIPEDIA, 2007. Android. [Consulté le 29.08.2016]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Android

Annexe 1 : Table ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source : ASSI TABLE, 11 août 2016

Annexe 2 : Tableau alphabet Base64

Valeur	Codage	Valeur	Codage	Valeur	Codage	Valeur	Codage
0	000000 A	17	010001 R	34	100010 i	51	110011 z
1	000001 B	18	010010 S	35	100011 j	52	110100 0
2	000010 C	19	010011 T	36	100100 k	53	110101 1
3	000011 D	20	010100 U	37	100101 l	54	110110 2
4	000100 E	21	010101 V	38	100110 m	55	110111 3
5	000101 F	22	010110 W	39	100111 n	56	111000 4
6	000110 G	23	010111 X	40	101000 o	57	111001 5
7	000111 H	24	011000 Y	41	101001 p	58	111010 6
8	001000 I	25	011001 Z	42	101010 q	59	111011 7
9	001001 J	26	011010 a	43	101011 r	60	111100 8
10	001010 K	27	011011 b	44	101100 s	61	111101 9
11	001011 L	28	011100 c	45	101101 t	62	111110 - (minus)
12	001100 M	29	011101 d	46	101110 u	63	111111 _ (underline)
13	001101 N	30	011110 e	47	101111 v		
14	001110 O	31	011111 f	48	110000 w	(complément) =	
15	001111 P	32	100000 g	49	110001 x		
16	010000 Q	33	100001 h	50	110010 y		

Source : WIKIPEDIA, 11 août 2016

Annexe 3 : Extrait de la table « T1 »

ID	Nom	Statut	TempsExpiration	Quota
1	cscqinfo	1	00:10:00.0000000	50

Annexe 4 : Extrait de la table « T2 »

ID	IDApplicatic	IDAdherent	DateDebutSession	DateExpirationSession	DateFinSession	NombreAcces	StatutSession
467	1	9997	07.09.2016 13:29:57	07.09.2016 13:28:59	07.09.2016 13:30:24	1	0
466	1	9997	07.09.2016 13:23:32	07.09.2016 13:22:34	07.09.2016 13:27:10	1	0
465	1	9997	07.09.2016 13:21:08	07.09.2016 13:19:28	07.09.2016 13:21:43	4	0
464	1	9997	07.09.2016 13:19:58	07.09.2016 13:19:01	07.09.2016 13:20:57	1	0
463	1	9997	07.09.2016 13:14:25	07.09.2016 13:15:27	07.09.2016 13:19:05	1	0
462	1	9997	07.09.2016 13:07:59	07.09.2016 13:08:01		1	1
461	1	9997	07.09.2016 11:56:28	07.09.2016 12:06:49	07.09.2016 13:04:00	4	0
460	1	9997	07.09.2016 11:45:46	07.09.2016 12:04:51		6	1
459	1	9997	02.09.2016 13:02:52	02.09.2016 13:03:06	02.09.2016 13:18:54	0	0
458	1	9997	02.09.2016 11:30:49	02.09.2016 11:50:53	02.09.2016 11:46:13	13	0
457	1	9997	02.09.2016 11:30:48	02.09.2016 11:40:52		1	1
456	1	9997	02.09.2016 11:22:46	02.09.2016 11:33:33		6	1
455	1	9997	02.09.2016 10:06:34	02.09.2016 10:16:36	02.09.2016 11:22:28	1	0
454	1	9997	02.09.2016 08:59:35	02.09.2016 09:10:04		3	1
453	1	9997	02.09.2016 08:49:49	02.09.2016 09:00:51	02.09.2016 08:51:32	7	0
452	1	9997	02.09.2016 08:37:56	02.09.2016 08:47:58		1	1
451	1	9997	02.09.2016 08:36:34	02.09.2016 08:46:36		1	1
450	1	9997	02.09.2016 08:32:33	02.09.2016 08:42:35		1	1
449	1	9997	02.09.2016 08:27:42	02.09.2016 08:38:10	02.09.2016 08:28:30	3	0
448	1	9997	02.09.2016 08:25:16	02.09.2016 08:35:18		1	1
447	1	9997	02.09.2016 08:24:11	02.09.2016 08:34:13		1	1
446	1	9997	02.09.2016 08:22:07	02.09.2016 08:32:10		1	1
445	1	9997	01.09.2016 16:52:13	01.09.2016 17:02:35		2	1
444	1	9997	01.09.2016 16:20:15	01.09.2016 16:55:15	01.09.2016 16:50:48	100	0
443	1	9997	01.09.2016 14:38:45	01.09.2016 14:59:53		1	1
442	1	9997	01.09.2016 14:12:52	01.09.2016 14:22:52	01.09.2016 14:16:14	0	0
441	1	9997	01.09.2016 13:47:08	01.09.2016 13:57:08		0	1

Annexe 5 : Extrait de la table « T3 »

ID	IDSession	ServiceInvoque	ServiceParametres	IDDemandeur	Date
1795	467	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:29:59
1794	466	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:23:34
1790	465	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:21:09
1791	465	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:21:21
1792	465	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:21:25
1793	465	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:21:28
1789	464	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:20:01
1788	463	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:14:27
1787	462	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 13:08:01
1783	461	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 11:56:31
1784	461	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	07.09.2016 11:56:38
1785	461	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	07.09.2016 11:56:44
1786	461	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	07.09.2016 11:56:49
1777	460	PS:ws_ListeAdherentGE	M=0	9997	07.09.2016 11:45:49
1778	460	PS:ws_LectureInfoAdher	M=0	9997	07.09.2016 11:46:05
1779	460	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	07.09.2016 11:46:21
1780	460	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	07.09.2016 11:54:18
1781	460	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	07.09.2016 11:54:31
1782	460	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	07.09.2016 11:54:51
1774	459	PS:ws_ListeAdherentGE	M=0	9997	02.09.2016 13:02:53
1775	459	PS:ws_ListeAdherentGE	M=0	9997	02.09.2016 13:03:01
1776	459	PS:ws_ListeAdherentGE	M=0	9997	02.09.2016 13:03:06
1761	458	PS:ws_ListeAdherentGE	M=0	9997	02.09.2016 11:30:52
1762	458	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	02.09.2016 11:31:01
1763	458	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	02.09.2016 11:31:11
1764	458	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	02.09.2016 11:31:17
1765	458	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	02.09.2016 11:31:24
1766	458	PS:ws_LectureInfoSouscr	M=0;AB=AM;SC=1	9997	02.09.2016 11:36:55
1767	458	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	02.09.2016 11:36:59
1768	458	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	02.09.2016 11:37:06
1769	458	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	02.09.2016 11:37:10
1770	458	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	02.09.2016 11:37:19
1771	458	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	02.09.2016 11:37:24
1772	458	PS:ws_LectureInfoSouscr	M=0;AB=4CH12;SC=40	9997	02.09.2016 11:37:29
1773	458	PS:ws_LectureInfoAbonn	M=0;AB=50	9997	02.09.2016 11:40:53
1760	457	PS:ws_ListeAdherentGE	M=0	9997	02.09.2016 11:30:52
1754	456	PS:ws_ListeAdherentGE	M=0	9997	02.09.2016 11:22:47
1755	456	PS:ws_LectureInfoAdher	M=0	9997	02.09.2016 11:22:57
1756	456	PS:ws_LectureInfoAdher	M=0	9997	02.09.2016 11:23:19
1757	456	PS:ws_LectureInfoAdher	M=0	9997	02.09.2016 11:23:24
1758	456	PS:ws_LectureInfoAdher	M=0	9997	02.09.2016 11:23:29