

Résumé

Ce Travail de Fin d'études est issu du stage que nous avons réalisé au sein du Centre d'Informatique pour la Région Bruxelloise (CIRB).

En effet, l'intégrateur de services régional FIDUS est parmi les infrastructures solutions faites par cet organisme. Il s'agit d'une plateforme informatique créée pour favoriser les échanges électroniques de données provenant de sources authentiques fédérales et régionales entre administrations de la région Bruxelloise, et ce, en toute sécurité et dans le respect de la vie privée.

Toute transaction (requête/réponse) est conservée dans un *audit log* permettant de tracer « qui, a fait quoi, quand et pourquoi il l'a fait ». Actuellement, FIDUS crée 8.000.000 d'enregistrement par an.

Notre projet avait comme objectif principal la réalisation de l'archivage de cet *audit log*, précisément les transactions qui datent plus qu'un an, mais encore de sécuriser les différents archives puisqu'elles contiennent des données à caractère personnel. Cela est fait par le biais d'une application *Spring Batch* avec *Spring Boot*.

La consultation des archives quant à elle devait être réalisée vu que le DPO, l'administrateur ou le CISO peuvent avoir besoin d'une information archivée dans les dix ans qui viennent. C'est pourquoi une deuxième application *web Spring Boot* a été indispensable.

Cette application présente deux interfaces. La première, interface login permet uniquement aux utilisateurs avec un droit d'accès de l'utiliser. La seconde interface permet aux utilisateurs de faire des recherches concernant des données précises et dans des fichiers archivés. Ils peuvent également via cet interface télécharger l'archive contenant la donnée voulue. Seule condition, deux attributs doivent être cryptés.

Par le présent rapport, nous présentons le CIRB avec ses missions et ses partenariats. Nous exposons également l'analyse fonctionnelle et technique aboutissant au choix de différentes technologies. Nous finirons par détailler les résultats finaux tout en exposant une annexe.

Un manuel d'emploi sera également détaillé. Le manuel correspond aux logiciels requis et aux configurations nécessaires afin que le *team Fidus* puisse utiliser ce travail et l'intégrer avec ses projets dans le futur.

Abstract

This end of study report finalizes of my internship at the **Brussels Regional Informatics Centre (BRIC)**.

Indeed, the regional service integrator FIDUS is among the infrastructure solutions made by this organization. It's an IT platform created to facilitate the electronic exchange of data from authentic federal and regional sources between administrations of the Brussels region, in a safe and respectful private life.

Any transaction (query / response) is kept in an audit log to trace "who did what, when and why it did". Currently, FIDUS creates 8,000,000 registrations a year.

Our project had as main objective the realization of the archiving of this audit log, precisely the transactions which date more than one year, but also to secure the various archives since they contain personal data. This is done through a Spring Batch app with Spring Boot.

The consultation of the archives is to be carried out because it's feasible that the DPO, the administrator or the CISO need an archived information in the ten years after. That's why a second Spring Boot web application was essential.

This application has two interfaces. The first, login interface only allows users with access rights to use it. The second interface allows users to search for specific data and archived files. They can also via this interface download the archive containing the desired data. Only condition, two attributes must be encrypted.

Through this report, we present the BRIC with its missions and partnerships. We also expose the functional and technical analysis leading to the choice of different technologies. We will finish by detailing the final results and exposing an appendix.

An employment manual will also be detailed. The manual corresponds to the required software and the necessary configurations so that the Fidus team can use this work and integrate it with its projects in the future.

Remerciements

Mes premiers remerciements vont à mon frère **Abdelfatah** qui sans ses encouragements et surtout son aide financier je n'aurais pas pu venir en Belgique afin de réaliser mon rêve de devenir ingénieur industriel dans le domaine qui me passionne.

Ensuite, je tiens à remercier mon promoteur académique, **GIOT Rudi** de l'Institut Supérieur Industriel de Bruxelles pour ses conseils avisés et le suivi qu'il a apporté à mon stage ainsi que pour son expérience dans la rédaction de ce document.

Je souhaite également remercier tout le *team Fidus* ainsi que le *team IAM* pour leur accueil et leur convivialité durant toute la durée de mon stage.

Plus précisément ma reconnaissance s'adresse à :

- **VANDERBORGHT Céline**, pour mon recrutement au sein de CIRB.
- **DE PESSEMIER Johan**, responsable de la conception et de la gestion des solutions et chef de service Fidus.
- **LE GRELLE Dominique**, mon promoteur industriel et le chef de projet qui m'a permis de réaliser ce stage et qui a pu superviser ce travail tout au long sur stage.
- **HACHET Serge**, l'architecte, qui a suivi l'évolution du projet et qui n'a pas cessé de m'orienter afin de mener à bien ce projet.

Table Des Matières

Résumé

Abstract

Remerciements

Liste des figures

I.	Introduction.....	1
1.1	Présentation de la société	1
1.2	Présentation de la plateforme FIDUS.....	2
1.3	Objectifs et cahier des charges	3
1.3.1	Contexte	3
1.3.2	Périmètre.....	6
1.4	Diagramme de GANTT	7
1.5	Méthodologie de travail.....	9
II.	Architecture applicative	10
2.1	Architecture générale.....	10
2.2	Architecture REST	10
2.3	Batch job.....	11
2.4	Serveur Solr	13
III.	Analyse fonctionnelle et technique.....	13
3.1	Diagramme de cas d'utilisation global	14
3.2	Diagrammes de classe	15
3.2.1	Archivage.....	15
3.2.2	Consultation des archives.....	16
IV.	Réalisation des deux applications	17
4.1	Environnement de travail.....	17
4.2	Frameworks.....	18
4.3	Implémentation.....	19
4.3.1	Tasklets, steps and ItemWriter.....	19
4.3.2	Logging in Spring Batch	20
4.3.3	Expressions Lambda	21
4.3.4	Cryptage	22
4.3.5	Arborescence des deux applications.....	24

4.3.6	Service RESTful	27
4.4	Résultats	28
4.4.1	Application d'archivage	28
4.4.2	Application de consultation.....	29
V.	Conclusion et perspectives.....	32
VI.	Bibliographie.....	34
VII.	Manuel d'utilisation	36
VIII.	Annexes	38
9.1	Méthodes Agiles et Scrum.....	38
9.2	Indexation des archives sous Solr	40
IX.	Glossaire	41

Liste des figures

Figure 1- Logo de CIRB, IRISteam et IRISNET	1
Figure 2- Schéma explicatif d'un audit log	3
Figure 3- Chaîne de partenaires	4
Figure 4- Tables existantes de la base de données	6
Figure 5- Diagramme statique général.....	6
Figure 6- Diagramme de GANTT	8
Figure 7- Architecture générale des deux applications.....	10
Figure 8- Batch job	12
Figure 9- Bean archiverJob, appel à toute étape	12
Figure 10- Diagramme des cas d'utilisation global	15
Figure 11- Diagramme de classe d'archivage.....	16
Figure 12- Diagramme de classe de la consultation.....	16
Figure 13- Tasklets principaux.....	19
Figure 14- Etapes pour chacune de tasklet	20
Figure 15- Création du fichier Json.....	20
Figure 16- Logging	20
Figure 17- Expression Lambda.....	21
Figure 18- Exemple sans expressions Lambda	22
Figure 19- Exemple avec expressions Lambda	22
Figure 20- Cryptage	23
Figure 21- Cryptage de seulement deux champs	23
Figure 22- Documentation de service RESTful	27
Figure 23- Test unitaire	27
Figure 24- Schéma JSON.....	28
Figure 25- Interface Login.....	29
Figure 26- Interface Recherche	29
Figure 27- Exemple de recherche.....	30
Figure 28- Documentation RESTful	31
Figure 29- Démarrage de serveur Solr	36
Figure 30- Création d'un core.....	36
Figure 31- Commande installation package	36
Figure 32- Fichier de configuration archivage.....	37
Figure 33- Chemin de la clé de cryptage	37

Figure 34- Fichier de configuration consultation	37
Figure 35- Processus Scrum.....	38
Figure 36- Stockage sous Solr	40

I. Introduction

1.1 Présentation de la société

Ce Travail de Fin d'études se déroule au Centre d'Informatique pour la Région Bruxelloise (CIRB), riche de quelque 400 collaborateurs, est l'organisme d'intérêt public qui, en Région de Bruxelles-Capitale, entend devenir le partenaire technologiquement neutre, compétitif, fiable et de qualité de toute institution publique qui souhaite, en connaissance de cause et de manière pro-active, introduire des TIC novatrices et cohérentes afin de maximiser d'une part, l'efficacité de son fonctionnement et d'autre part, la convivialité des services aux Bruxellois, aux entreprises et aux visiteurs.

Le CIRB a été créé par la loi du 21 août 1987, modifiée par l'ordonnance du 20 mai 1999 portant sur sa réorganisation ainsi que par l'ordonnance du 29 mars 2001. L'ordonnance du 8 mai 2014 le consacre comme intégrateur de services régional.

CIRB est chargé de toute mission de développement et d'assistance informatique, télématique et cartographique à l'égard:

- Des institutions publiques régionales, locales et communautaires ;
- Des cabinets des Ministres et Secrétaires d'Etat du Gouvernement de la Région de Bruxelles-Capitale ;
- Des organismes d'intérêt public de la Région de Bruxelles-Capitale.



Figure 1- Logo de CIRB, IRISteam et IRISNET

Le CIRB rassemble l'ensemble des pouvoirs locaux et régionaux autour de projets de développement IT, d'infrastructure mutualisée, de télécommunication ainsi que de mise à disposition des ressources humaines à travers l'Asbl IRISteam.

Le CIRB est aussi membre fondateur et actionnaire de la sclr IRISnet, dont la mission est la gestion et le développement de l'infrastructure réseau de télécommunication à large bande de la Région de Bruxelles-Capitale. Le réseau IRISnet couvre toute la Région avec plus de 380 km de câblage de fibres optiques.

1.2 Présentation de la plateforme FIDUS



Avant la création de FIDUS, l'échange des données entre les institutions fédérales se faisait par l'intermédiaire de la Banque Carrefour de la Sécurité Sociale « BCSS ». Puis, un intégrateur FEDICT est mis en place, Service public FÉDÉral Technologie de la Communication et de l'Information, qui a changé de nom après un an vers « BOSA ». Et suite à la création des institutions régionales, ils ont pensé à concevoir des intégrateurs de service en Flandre prenant le nom de « MAGDA ». De même, ils ont créé en Wallonie BCED, la Banque Carrefour d'Echange des Données.

En 2014, ils ont constaté qu'il manque un intégrateur de service régional sur la région Bruxelloise, donc en vertu de l'ordonnance du 8 mai 2014, toute institution publique Bruxelloise qui le souhaite peut profiter de l'intégrateur de services régional Fidus et c'est le CIRB qui assure ce rôle. Du coup, tout échange de données devait passer par cet intégrateur d'où vient la naissance de la plateforme FIDUS qui est le nom donné à cet intégrateur, la plateforme informatique spécifique a été mise en place pour mettre en œuvre toutes les composantes nécessaires à ces échanges : cryptage, signature électronique, contrôle des autorisations par une banque de règles, conservation des traces.

Depuis début novembre 2015, ils étaient prêts et en juin 2016 FIDUS a été commencée d'être utilisée.

1.3 Objectifs et cahier des charges

1.3.1 Contexte

Un *audit log* contenant des données à caractère personnel est la solution utilisée pour répondre à la question suivante:

Qui a accédé à **quelle donnée personnelle privée** concernant **qui** en utilisant **quelle finalité** et **quand**.

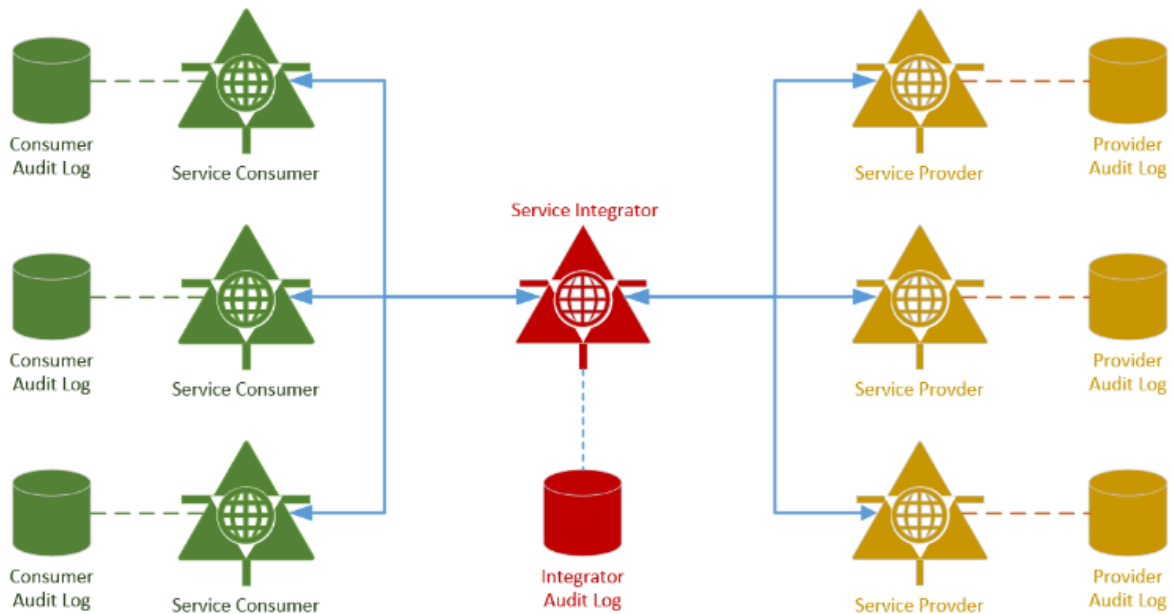


Figure 2- Schéma explicatif d'un audit log

En pratique, nous pouvons traduire cela par les actions suivantes :

- Le client va uniquement enregistrer **la demande** et le contexte de la demande. Aucune copie des données contenues dans la source authentique n'est conservée.
- L'intégrateur (s) enregistre uniquement les **métadonnées** nécessaires pour lier les messages du client et du fournisseur.
- Les fournisseurs consigneront uniquement **la réponse**. Ils gèrent déjà la source authentique et possèdent les données sensibles.
- **La combinaison** des logs des clients, des intégrateurs et des fournisseurs peut être demandée pour reconstruire l'intégralité de l'audit log.

Au minimum, les informations suivantes doivent être enregistrées sur chaque partenaire de chaîne :

Pour le client, les « personnes qui demandent » et « à propos de qui » seraient idéalement des numéros nationaux.

- | | | |
|---|--|---|
| <ul style="list-style-type: none"> • Who Requests • About Whom • Which Finality • Consumer Message ID • Timestamp • Request message | <ul style="list-style-type: none"> • Consumer ID • Provider ID • Consumer Message ID • Provider Message ID • Finality • Timestamps | <ul style="list-style-type: none"> • Integrator ID • Provider Message ID • Timestamp • Response message |
|---|--|---|



Figure 3- Chaîne de partenaires

Il y a deux ans que la plateforme Fidus est mise en place et ils ont déjà eu 8.000.000 de transactions et vu que seuls les logs les plus récents peuvent être conservés dans une base de données en ligne, un archivage devient indispensable. Du coup, nous avons pensé à stocker les anciens logs dans des fichiers de format JSON crypté.

La base de données *audit log* est une base de données PostgreSQL servant à garder en mémoire toutes les transactions faites par les institutions publiques Bruxelloises.

Nous y trouvons deux tables « consumer » et « provider » qui ont les mêmes attributs et qui représentent le même type d'information, les voilà :

id: identifiant de chaque enregistrement de la base de données.

transaction_id: *Univeral Unique IDentifier* UUID, qui représente le numéro de transaction unique.

internal_message_id: quand la demande se fait par le consumer, un identifiant message lui sera fourni pour récupérer la réponse quand elle sera disponible.

external_message_id: le identifiant de message de Fidus aux institutions.

external_timestamp: c'est l'heure à laquelle le consumer a envoyé la requête.

external_message_context: faire la liaison entre une donnée business (person ID par exemple) et l'audit log.

application_id: CN de certificat que le consumer utilise pour avoir la réponse.

institute: le nom de l'institution.

legal_context: ou encore finalité, la raison pour laquelle la demande a été faite.

endpoint: l'adresse de web service appelé.

action : ce que le consumer peut faire quoi comme opération *WSDL*, fermé ou reporté un dossier, changer un service, etc.

request: la demande du client.

response: la réponse du fournisseur.

error: erreurs liées à la base de données, la validation, accès à certaines données, etc.

request_timestamp: l'heure à laquelle Fidus a reçu la demande.

responseTimestamp: l'heure à laquelle Fidus a envoyé la réponse.

keyType : représente le registre national.

keyValue: la donnée à caractère personnel, qui pourrait représentée le Numéro de Registre National d'une personne.

Voici les tables qui construisent la base de données :

Provider	Consumer
+ action: text	+ action: text
+ application_id: text	+ application_id: text
+ end_point: text	+ end_point: text
+ error: text	+ error: text
+ external_message_id: text	+ external_message_id: text
- id: text	- id: text
+ institute: text	+ institute: text
+ internal_message_id: int	+ internal_message_id: text
- key_type: text {id}	+ Key_type: text {id}
- key_value: text	+ key_value: text
+ legal_context: text	+ legal_context: text
+ request: text	+ request: text
+ request_timestamp: text	- request_timestamp: text
+ response: text	+ response: text
+ response_timestamp: text	+ response_timestamp: text
+ transaction_id: text	+ transaction_id: text

Figure 4- Tables existantes de la base de données

Par la suite, nous allons mettre à disposition de certains utilisateurs la possibilité de faire des recherches sur les archives mais également de télécharger les différentes archives.

1.3.2 Périmètre

L'application de consultation sera utilisée par trois utilisateurs qui sont l'administrateur, le CISO et le DPO. Ils vont consulter les archives en faisant des recherches s'ils ont besoin d'une donnée archivée à caractère personnel dans les dix ans qui viennent.

Ceci est illustré dans le diagramme statique suivant :

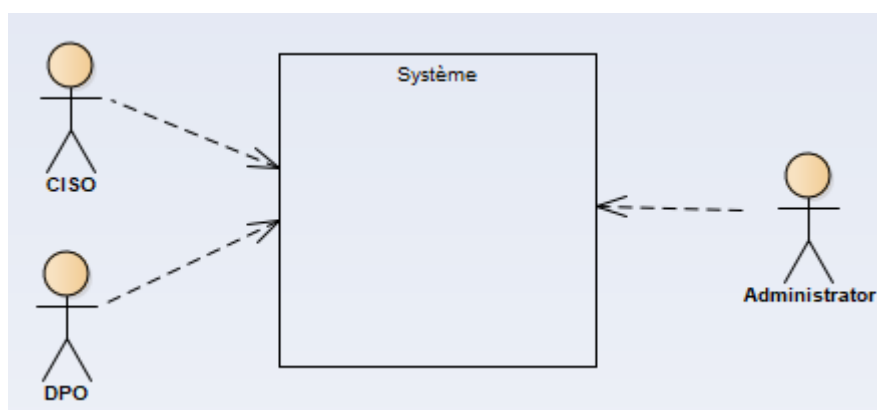


Figure 5- Diagramme statique général

1.4 Diagramme de GANTT

La suivante illustration de diagramme de GANTT montre la répartition temporelle des tâches que nous avons menées à bien au cours de notre stage. Le projet peut être décomposé en six grandes tâches chacune avec sa date d'échéance.

Les deux premières tâches consistaient à faire une analyse détaillée sur les attentes du projet ainsi que sur le choix des technologies en veillant à travailler avec celles les plus demandées dans le marché du travail.

La troisième tâche était l'extraction des données qui datent plus qu'un an sous format *JSON*.

La quatrième s'occupait du cryptage de cet archive en implémentant deux solutions et en stockant la clé de cryptage par le biais de l'API *Java KeyStore*.

Une cinquième tâche consistait à indexer l'archive sous Solr, et la dernière reposait sur la consultation de ces archives.

























		Nom	Durée	Début	Fin	Prédécesseurs
1		<input type="checkbox"/> Analyse fonctionnelle	11 jours	17/08/18 9:00	3/09/18 9:00	
2		lire la documentation sur Confluence	3 jours	17/08/18 9:00	22/08/18 9:00	
3		Diagrammes UML	7 jours	23/08/18 9:00	3/09/18 9:00	
4		<input type="checkbox"/> Analyse technique	11 jours	24/08/18 9:00	10/09/18 9:00	
5		Définition des premières tâches	6 jours	24/08/18 9:00	3/09/18 9:00	
6		analyse, recherche et choix de technologies	5 jours	3/09/18 9:00	10/09/18 9:00	3
7		<input type="checkbox"/> Extraction des données	13 jours	11/09/18 9:00	28/09/18 9:00	1;4
8		Générer un projet Spring boot avec toutes les dépendances nécessaires	3 jours	11/09/18 9:00	14/09/18 9:00	
9		Lire les données qui datent plus qu'un an d'une BD Postgre et les écrire dans un fichier JSON	9 jours	17/09/18 9:00	28/09/18 9:00	
10		<input type="checkbox"/> Cryptage	34 jours?	1/10/18 9:00	16/11/18 9:00	
11		<input type="checkbox"/> Sécuriser l'archive avec les algorithmes RSA et AES	34 jours?	1/10/18 9:00	16/11/18 9:00	
12		Première solution	5 jours?	1/10/18 9:00	8/10/18 9:00	
13		Deuxième solution	2 jours	22/10/18 9:00	24/10/18 9:00	17
14		Troisième solution	10 jours	25/10/18 9:00	8/11/18 9:00	
15		<input type="checkbox"/> Clé et tests	5 jours	9/11/18 9:00	16/11/18 9:00	14
16		Mettre la clé de cryptage avec l'API Java Key Store	5 jours	9/11/18 9:00	16/11/18 9:00	
17		<input type="checkbox"/> Solr server	8 jours	9/10/18 9:00	19/10/18 9:00	12
18		Indexer l'archive sur Solr afin de le consulter	8 jours	9/10/18 9:00	19/10/18 9:00	
19		<input type="checkbox"/> Consultation	14 jours	19/11/18 9:00	7/12/18 9:00	
20		Elaborer un service Restful qui permet de rechercher et télécharger les archives	6 jours	19/11/18 9:00	27/11/18 9:00	
21		Créer une IHM en Angular6 qui communique avec le service de recherche	7 jours	28/11/18 9:00	7/12/18 9:00	
22		Documenter l'application	4 jours	10/12/18 9:00	14/12/18 9:00	1;4;7;10;17
23		Présentation devant mon chef de projet et le chef du service	1 jour	7/01/19 9:00	8/01/19 9:00	
24		Transférer les connaissances (sous demande de mon promoteur industriel)	1 jour	8/01/18 9:00	9/01/18 9:00	

Figure 6- Diagramme de GANTT

1.5 Méthodologie de travail

Concernant notre méthodologie de travail tout au long du stage, nous avons appliqué la méthode Scrum à travers l'outil **JIRA software**. Ce dernier est un logiciel de gestion de projet Agile, développé par Atlassian, qui prend en charge toute méthodologie Agile.

Et pour ce qui concerne *Scrum*, c'est une méthode Agile où les produits sont conçus en une série d'itérations à durée fixe. Elle s'appuie sur les quatre piliers suivants : la planification des sprints, des stand-ups deux fois par semaine, des démonstrations de sprint et des rétrospectives.

Ainsi, nous organisons des réunions hebdomadaires de planification au cours desquelles nous réalisons des sprints ayant des durées entre deux à trois semaines. Chaque *sprint* comportait des tâches attribuées à chacun que nous évaluons en fonction du niveau de difficulté.

À la fin de chaque *sprint*, nous organisons un *Sprint review* afin de faire une démonstration du travail effectué.

Comme nous travaillons à plusieurs sur la plateforme Fidus, nous avons dû utiliser un outil permettant la gestion de versions de projets. Pour cela nous utilisons **Bitbucket** qui nous permet de gérer nos dépôts *Git*.

Tout notre travail se reposait sur un système de branches propre au modèle *Git* permettant d'isoler certaines fonctionnalités du reste du développement d'une part et d'une autre part de garder une trace de ce que nous avons fait tout au long du projet. Nous avons deux branches sur lesquelles nous trouverons l'application de l'extraction et l'archivage des données et une autre branche où la deuxième application est mise.

II. Architecture applicative

2.1 Architecture générale

Le *back-end* de l'application de l'extraction et d'archivage des données a été développé en *Java* avec le *framework Spring boot* et déployée sur le serveur *Tomcat* qui vient avec ce dernier.

Et pour ce qui est application de consultation, le *back-end* est développé en *Spring boot* aussi et l'application *front-end* a été faite en *Angular6*. Le choix de cette technologie impose une architecture où le client et le serveur ne sont pas dépendant. La communication entre ces deux applications (*back-end* et *front-end*) se fait via des services web *Restful*.

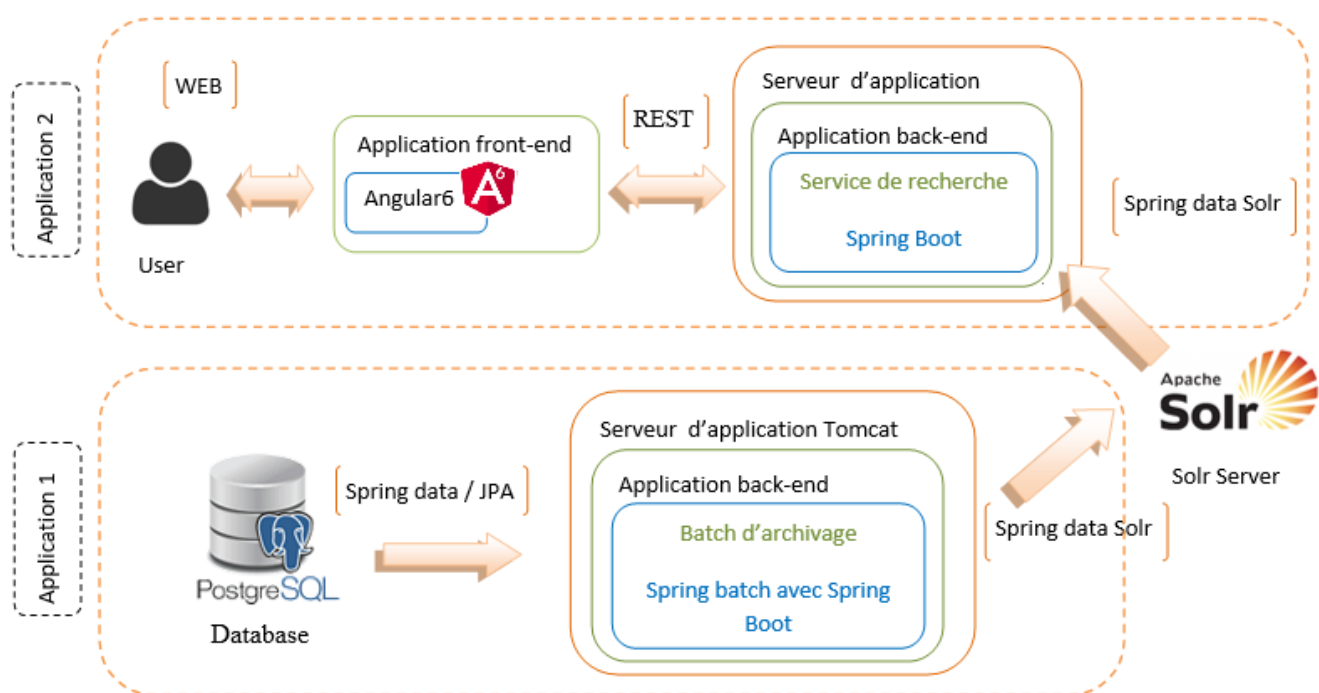


Figure 7- Architecture générale des deux applications

2.2 Architecture REST

Rest, **RE**presentational **State** **T**ransfer, est un style d'architecture *Web* permettant de construire des services web. C'est une architecture orientée ressource, c'est-à-dire une architecture permettant de rendre les ressources d'un système accessible. Les ressources modélisant les données et les fonctionnalités d'un système. Ainsi, une ressource peut être une base de données, un fichier, ou n'importe quel objet auquel un service peut y accéder.

REST repose sur le protocole *HTTP* plutôt que de réinventer une surcouche. On accède à une ressource par son *URI* pour procéder aux diverses opérations supportées par *http* (GET, POST, PUT, DELETE, etc.).

REST est souvent préféré à *SOAP*, *Simple Object Access Protocol*, pour sa légèreté et sa facilité à mettre en place. Contrairement de *SOAP* qui ne supporte que le format XML, bien que de complexes protocoles lui permettent de supporter d'autres formats, *REST* n'impose aucun format d'échange. On est donc libre de représenter les données en XML, JSON, PHP ou tout autre langage.

Cependant, *REST* étant un style d'architecture et non un standard, chacun crée sa propre architecture en fonction des critères de conception retenus.

2.3 Batch job

Un Batch job lit, traite et écrit les données sous un format spécifique:



Nous avons le *JobRepository* qui est connecté au *JobLauncher*, *Job* et aux *Step*. Ces dernières sont connectées elles-mêmes à ces trois éléments : *ItemReader*, *ItemProcessor* et *ItemWriter*.

JobLauncher : exécute les jobs en lisant les paramètres de *Job* et d'autres informations existant dans le *JobRepository* qui est persisté dans un système de gestion de base de données.

JobRepository : fournit les opérations *CRUD* pour le *JobLauncher* et pour les autres mécanismes *batch core* afin de persister leur état durant l'exécution.

Job : avant de le lancer, le *JobLauncher* cherche tous les paramètres liés au *Job* du *JobRepository*. Le *job* peut contenir minimum un *Step* ou plusieurs afin de le lancer comme un *Batch job* et puis il va être exécuté comme un *Step*.

Step : représente un morceau de code, par exemple, la lecture des données d'un fichier, l'envoi d'un mail sont considérées comme des étapes d'un *Batch job*.

ItemReader : lit les données entrantes et partage chaque élément soit avec le *ItemReader* ou encore avec le *ItemProcessor*.

ItemWriter : écrit les données d'un élément dans un *output*.

ItemProcessor: traite les données entrantes de *ItemReader* et les transforme à une forme que le *ItemWriter* peut comprendre.

Pour chaque Step nous configurons l'*ItemReader*, *ItemWriter*. L'*ItemProcessor* est optionnel. Nous signalons que nous avons des interfaces qui aident à les configurer.

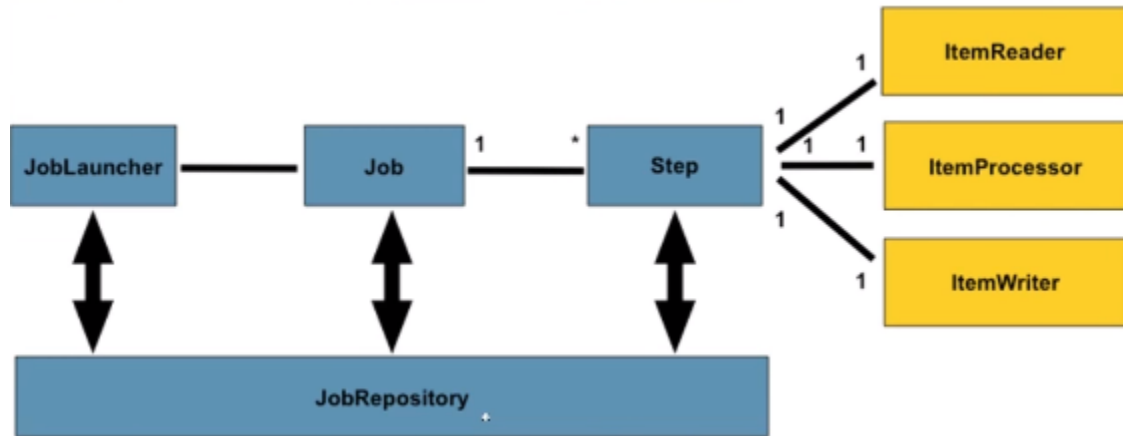


Figure 8- Batch job

Idéalement, nous avons travaillé avec un seul job contenant deux *Steps*. Au démarrage de l'application, la première étape qui s'exécute est celle de l'extraction des données qui datent plus qu'un an sous format *JSON* avec cryptage de ce dernier et la deuxième étape est responsable sur l'indexation du fichier crypté sous *Solr* afin de le récupérer pour faire des recherches par après.

```
@Bean
public Job archiverJob() throws NoSuchAlgorithmException {
    return jobBuilderFactory.get("archivingJob")
        .incrementer(new RunIdIncrementer())
        .start(fieldsEncryptionStep())
        .next(solrStep())
        .build();
}
```

Figure 9- Bean archiverJob, appel à toute étape

Chaque *step* sera détaillée dans les parties qui suivent.

2.4 Serveur Solr

Nous avons extrait les données sous format *CSV* mais vu que le « split » peut faire partie de la clé de cryptage, ce qui va engendrer un problème précisément quand nous allons vouloir décrypter afin de consulter les archives. De ce fait, nous avons choisi de travailler avec des fichiers *JSON*.

Après l'extraction des données sous fichier *JSON*, nous devons mettre les données dans un endroit afin de les consulter. En conséquent, nous étions face à trois solutions à mettre en place :

1. Soit garder les données extraites dans une autre BD *Postgre*, ce qui n'était pas trop intéressant pour nous en tant que stagiaire en fin d'année d'études d'ingénieur où nous devons explorer d'autres technologies.
2. Soit utiliser une base de données orientée documents, *SGBD NoSQL* via *mongoDB*, mais vu qu'il n'est pas parmi les standards de logiciels à utiliser au sein de CIRB, ça n'a pas été acceptée par le *project analyst*.
3. Ou encore créer un objet sous la plateforme logicielle de moteur de recherche *Solr* contenant le contenu du fichier et ses métadonnées, et quand on veut le consulter on récupère l'objet et on recrée le fichier avec ses métadonnées.

Finalement, nous optons pour la dernière solution, qui sera exposée dans la partie d'implémentation.

III. Analyse fonctionnelle et technique

Avant d'entamer le développement des applications, nous avons dû faire une analyse de nos deux applications, et afin de les réaliser nous avons utilisé les différents diagrammes primordiaux pour notre projet que fournit le langage de modélisation UML pour représenter un logiciel à développer.

Par contre, étant donné qu'aucune des applications n'était mise en place par le team Fidus, il a fallu organiser des réunions avec le chef et l'architecte de projet afin de comprendre et transcrire les objectifs et les fonctionnalités qu'ils attendent de cet archivage.

L'analyse que nous exposons dans cette partie reprend seulement les diagrammes les plus nécessaires qui illustrent l'analyse de besoins de l'utilisateur et d'autres faisant partie de la conception de l'application.

3.1 Diagramme de cas d'utilisation global

Les besoins de l'utilisateur sont modélisés à l'aide de diagramme de cas d'utilisation. Ce dernier a pour but de donner une vision globale des fonctionnalités d'un système nécessaires aux utilisateurs. Il identifie les acteurs, les intervenants externes de système, qui interagissent avec un système à travers de cas d'utilisation qui représente une fonctionnalité. Il reprend donc toutes les fonctionnalités que doit fournir un système.

On a trois acteurs qui interviennent au niveau de l'application de consultation, l'administrateur qui gère les droits d'accès aux archives et consulte ce dernier, et un DPO qui fait juste des recherches sur les archives.

Les acteurs représentés sur notre diagramme de cas d'utilisation sont les suivants :

DPO : Data Protection Officer, c'est celui responsable sur l'ajout de certificats et la consultation des archives.

Consumer : Entité initiant un flux de communication en appelant un service à un moment donné. Cela peut être dû à une interaction humaine (end-user) ou à un processus automatisé.

Supplier : Entité qui permet d'autoriser l'accès aux données d'une source authentique.

Administrateur : Il configure les services en rajoutant un « legal context » qui se définit par la raison pour laquelle un *consumer* a demandé une information d'un *supplier*, il rajoute aussi le *consumer* et les filtres. En outre, il réalise les archives, gère les droits d'accès et consulte les archives.

CISO : Corporate Information Security Officer, qui répertorie les accès aux données qui sont conservées 1 an et archivées 10 ans. C'est lui qui a le droit de consulter l'audit log.

La figure ci-dessous illustre les cas des utilisations :

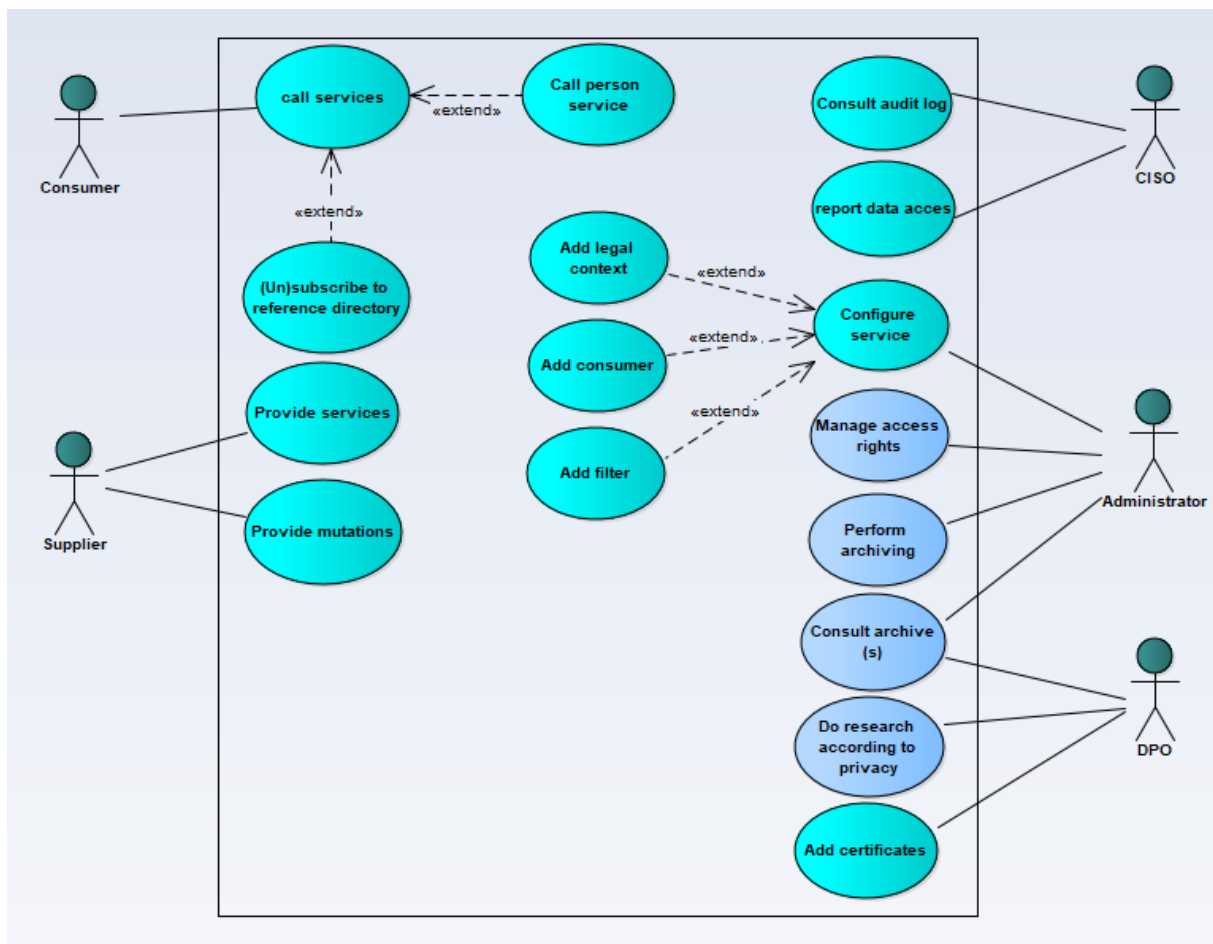


Figure 10- Diagramme des cas d'utilisation global

3.2 Diagrammes de classe

L'*audit log* est composé de deux tables une pour le *Provider* et une autre *Consumer*. Les entités qu'on aura besoin pour la mise en place de l'application sont reprises dans les diagrammes de classe suivants:

3.2.1 Archivage

En plus des deux tables *Provider* et *Consumer* détaillées dans le deuxième chapitre *Objectifs et cahier des charges*, nous rajoutons :

Une table *JsonArchive*, qui constitue d'un identifiant et de la date de l'archivage, puis trois actions qu'on peut faire sur ses tables. Après avoir les archiver, on les supprimera de l'*audit log* afin de le nettoyer et par la suite nous pouvons les consulter en effectuant des recherches. Dix ans plus tard, les archives seront supprimées définitivement.

Mais, encore nous avons dû utiliser une autre table qui sert à stocker le fichier extrait sous *Solr* ainsi que ses métadonnées, *SolrArchive*.

id: id du fichier

date: date de l'indexation du fichier sur Solr.

content: contenu du fichier en byte.

extension: que nous avons choisi de travailler avec « .JSON ».

fileName: header du fichier contenant le nom du fichier avec l'heure du stockage sur *Solr*.

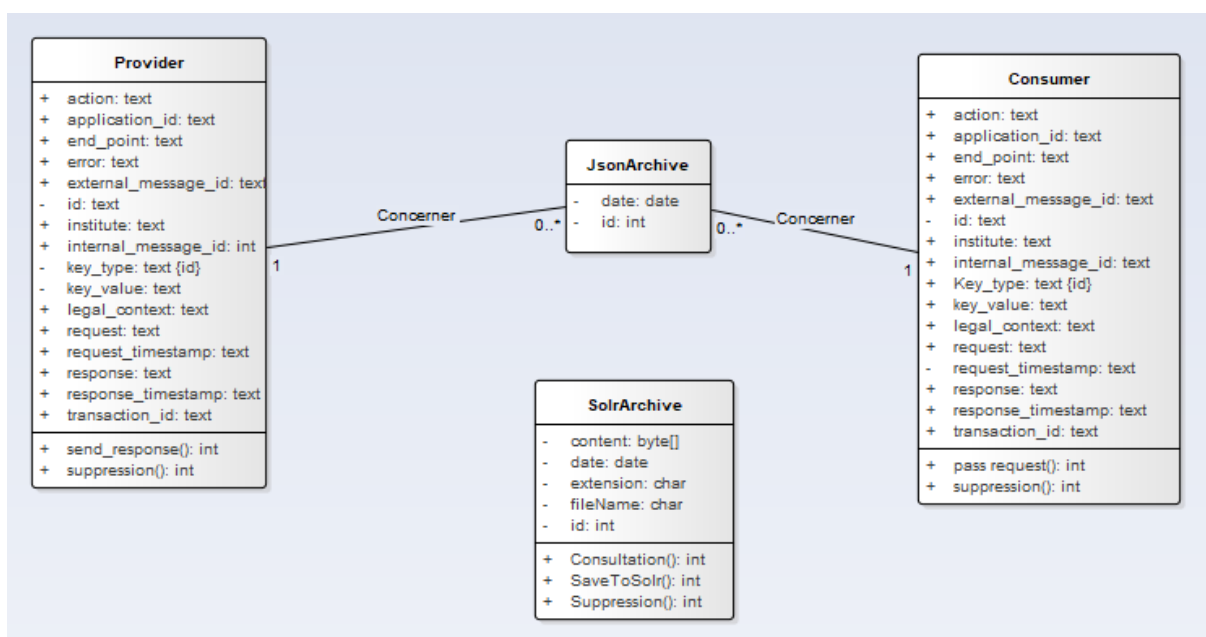


Figure 11- Diagramme de classe d'archivage

3.2.2 Consultation des archives

Chaque utilisateur a un rôle mais seul l'administrateur, CISO et le DPO qui peuvent le consulter. Le rôle a comme clé primaire « id », un attribut « code » qui est unique pour faire le traitement et le label qui sera soit *admin* pour l'administrateur, *ciso* pour le CISO ou encore *dpo* pour le DPO.

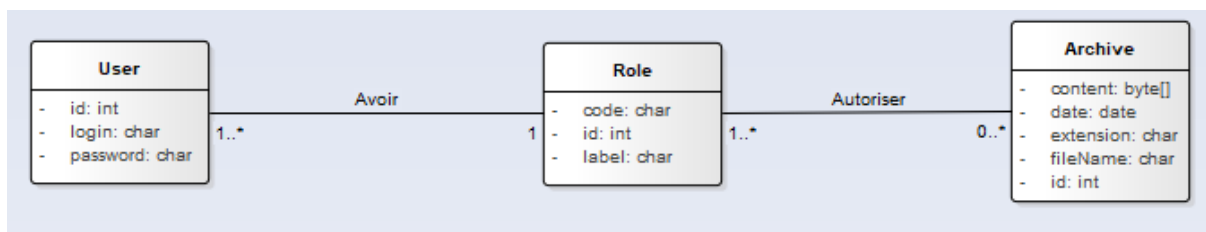


Figure 12- Diagramme de classe de la consultation

IV. Réalisation des deux applications

4.1 Environnement de travail

Le CIRB utilise le logiciel **Entreprise Architect** qui représente la solution idéale à l'échelle de l'entreprise pour visualiser, analyser, modéliser, tester et maintenir tous les systèmes, logiciels, processus et architectures.

Il utilise comme SGBD, **PgAdmin**, qui est une plateforme d'administration et de développement open source la plus populaire et riche en fonctionnalités pour PostgreSQL qui est un système de gestion de base de données relationnelle et objet (SGBDRO) Open Source.

L'application back-end est faite en **Java EE** sous l'IDE **eclipse** et déployée sur le serveur Tomcat qui est intégré par Spring boot, ce qui fait partie des avantages des applications Spring boot. Et pour ce qui est build et la gestion des dépendances nous avons utilisé Maven. Ce dernier, est un outil de construction de projet open source développé par la fondation Apache, initialement pour les besoins du projet Jakarta Turbine. Il permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet Java. Il permet entre autres d'automatiser certaines tâches (compilation, tests unitaires et déploiement de projet), de gérer les dépendances du projet et de générer des documentations concernant celui-ci.

L'application front-end a été développée en **Angular6**. Le choix de cette technologie impose une architecture où le client et le serveur ne sont pas dépendant. La communication entre ces deux applications (back-end et front-end) se fait via des services web Restful.

Et concernant la gestion des dépendances, on a opté pour l'utilisation de **NPM 'NodeJS Package Manager'**, c'est exactement la même chose que *maven* mais en front-end. C'est le gestionnaire de packages pour JavaScript et le plus grand registre de logiciels dans le monde qui facilite le développement.

Il m'a été demandé non seulement de réaliser les archives mais encore de les crypter vu qu'elles contiennent des données à caractère personnel (Numéro de Registre National ou toute autre donnée privée). Si bien que nous avons utilisé les **algorithmes AES+RSA** en java standard. Et pour faire la consultation des archives, il fallait mettre l'archive crypté dans un serveur. De ce fait, nous avons travaillé avec **Solr**. Ce dernier est une application serveur qui va permettre l'indexation de fichiers ou de bases de données. Il deviendra alors possible

d'effectuer des recherches à travers ces données. Solr intègre un système d'analyse syntaxique, issue de la technologie *Lucene*, qui donne toute la pertinence aux résultats des recherches.

Finalement, nous avons utilisé le logiciel **Java Key Store**, qui est un *Repository* de certificats de sécurité qui nous va permettre de stocker la clé de cryptage qui servira au décryptage lors de la consultation des archives.

4.2 Frameworks

Pour ce qui est frameworks, nous avons choisi de prendre en main et d'en utiliser plusieurs afin de faciliter le travail et d'aboutir aux résultats souhaités à savoir. **Spring Data JPA**, **Java Persistence API**, est l'un des projets de Spring qui propose une autre façon d'accéder aux données vu que L'implémentation d'une couche d'accès aux données, le pattern DAO, d'une application est une tâche fastidieuse. Ce framework vise à améliorer de manière significative la mise en œuvre des couches DAO en gérant la correspondance entre des objets d'une application et les tables d'une base de données nommée ORM, Object-Relational Mapping. JPA se repose principalement sur l'utilisation d'annotations qui permettent de définir des objets métier servant d'interface entre la base de données et l'application.

Nous avons utilisé le framework **Spring Batch** qui est un framework open source pour le traitement par lots. Il s'agit d'une solution légère et complète conçue pour permettre le développement d'applications par lots robustes.

Afin de faciliter le développement d'applications fondées sur Spring en offrant des outils permettant d'obtenir une application packagée en jar, totalement autonome. Nous avons utilisé le framework **Spring boot** Ce qui nous intéresse particulièrement, puisque nous essayons de développer des Micro services. À noter qu'il est toujours facile de déployer les applications **Spring Boot** sur une variété de plates-formes cloud, sur des images de conteneur, telles que Docker, ou encore sur des machines virtuelles ou/et réelles.

Pour gérer les droits d'accès à l'application de consultation des archives, nous choisissons de travailler avec **Spring Security**, qui est un Framework de sécurité léger fournissant une authentification et un support d'autorisation afin de sécuriser les applications Spring.

Nous avons utilisé le **JWT** en parallèle avec le **Spring Security** pour gérer l'authentification. Les JWT ou 'JSON Web Token' sont des jetons générés par un serveur lors de l'authentification d'un utilisateur sur une application Web, et qui sont ensuite transmis au client.

Finalement pour documenter les services *Restful*. Nous avons à notre disposition l'outil **Spring Rest Docs** qui combine une documentation écrite à la main écrite avec **Asciidoctor** et des extraits générés automatiquement avec **JUnit**. Il aide principalement à produire une documentation précise et bien structurée ce qui aidera mon team à comprendre bien le code source de ce travail.

4.3 Implémentation

Nous allons décrire les étapes de développement de composants cruciaux des trois applications à savoir l'archivage, consultation et l'application front-end.

4.3.1 Tasklets, steps and ItemWriter

Tasklets :

Nous avons fait l'archivage, le cryptage ainsi nous avons mis les archives cryptés sous *Solr* par le biais de deux grands *Tasklets*, à savoir :

Field encryption tasklet : cette étape est responsable sur l'extraction des données et sur le cryptage de ces deux données « legal_context » et « institute » seulement puisque seule c'est deux peuvent dévoiler la donnée à caractère personnel.

Et *Solr tasklet* : qui permet de sauvegarder le fichier généré par la *Field encryption tasklet* sur le serveur *Solr* afin de le consulter.

```
// Tasklets config

@Bean
protected Tasklet fieldsEncryptionTasklet() throws NoSuchAlgorithmException {
    return new FieldsEncryptionTasklet(consumerRepository, providerRepository, writer(), jksPassword, jksFilepath);
}

@Bean
protected Tasklet solrTasklet(SolrArchiveRepository solrArchiveRepository) {
    return new SolrTasklet(solrArchiveRepository, archiveDirectory);
}
```

Figure 13- Tasklets principaux

Steps:

Step *fieldsEncryptionStep* est responsable sur l'appel de *tasklet fieldsEncryptionTasklet*. Quant au *solrStep* qui fait l'appel du *tasklet solrTasklet*.

```
@Bean
protected Step fieldsEncryptionStep() throws NoSuchAlgorithmException {
    return stepBuilderFactory.get("fieldsEncryptionStep")
        .tasklet(fieldsEncryptionTasklet())
        .transactionAttribute(new DefaultTransactionAttribute(TransactionDefinition.PROPROPAGATION_NEVER))
        .build();
}

@Bean
protected Step solrStep() {
    return stepBuilderFactory.get("solrStep")
        .tasklet(solrTasklet(solrArchiveRepository)).build();
}
```

Figure 14- Etapes pour chacune de tasklet

ItemWriter :

Il est utilisé par la *FieldsEncryptionTasklet* pour construire le fichier avec un nom ayant un format bien spécifique « archive_date.json », et l'écrire sur le disque. Le chemin de stockage du fichier est stocké sous le fichier « application.yml » pour qu'il soit facilement modifiable.

```
@Bean
public FlatFileItemWriter<JsonArchive> writer() {
    FlatFileItemWriter<JsonArchive> writer = new FlatFileItemWriter<>();
    SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
    String date = sdf.format(new Date());
    writer.setResource(new FileSystemResource(archiveDirectory + "archive_" + date + ".json"));
    writer.setSaveState(true);
    writer.open(new ExecutionContext());
    writer.setLineAggregator(new ArchiveJsonItemAggregator());
    return writer;
}
```

Figure 15- Création du fichier Json

4.3.2 Logging in Spring Batch

Nous avons utilisé les logging qui nous permettent de voir ce qui se passe lors de l'exécution d'un job :

```
@Scheduled(cron = "${batch.archivingCron}")
public void launchMigration() throws JobExecutionAlreadyRunningException, JobRestartException,
    JobInstanceAlreadyCompleteException, JobParametersInvalidException {

    logger.info("***** Starting Archiving Job *****");

    JobParameters params = new JobParametersBuilder().addDate("date", new Date())
        .toJobParameters();
    jobLauncher.run(archiverJob, params);
    logger.info("***** Archiving Completed! *****");
}
```

Figure 16- Logging

4.3.3 Expressions Lambda

Étant donné que nous avons utilisé *Java8*, il était recommandé d'utiliser les expressions Lambda. Ces dernières sont parmi les nouveaux concepts introduits dans *Java 8*, elles ont connu un développement considérable pour ses rapports étroits avec les langages de programmation fonctionnelle. Son but primordial est de permettre de passer en paramètres un ensemble de traitements et son intérêt principal provient de la simplicité de sa syntaxe.

Comme particularité de ces expressions, elles font référence aux fonctions anonymes, au sens où la définition de celles-ci ne demande aucun nom. Du coup, elles sont définies directement là où elles sont appelées. Cela permet de simplifier la création des fonctions à la volée, pour un besoin spécifique sans autant créer plusieurs classes.

La seule condition pour que l'affectation d'une expression lambda à une variable (d'une interface fonctionnelle) soit possible est que, la signature de la méthode abstraite de l'interface fonctionnelle doit *matcher* celle de l'expression lambda.

On peut même imaginer que l'expression lambda est une implémentation de l'interface fonctionnelle. Une expression lambda se compose de trois parties:

- Paramètre(s)
- -> opérateur
- Corps (exécution)

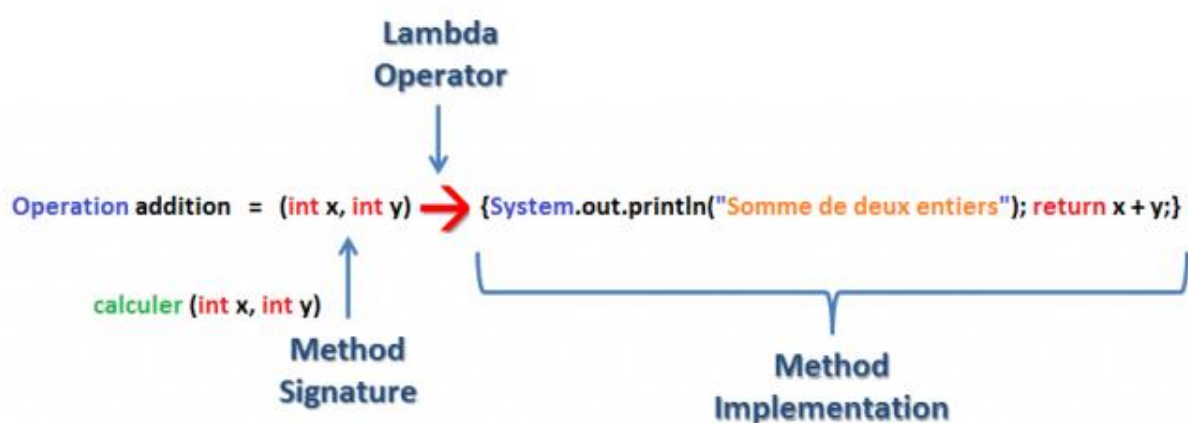


Figure 17- Expression Lambda

Elles sont donc une des plus importantes nouveautés de Java8 voire la plus importante évolution apportée langage Java depuis sa création.

Dans les figures qui suivent, un exemple de partie de code sans et avec ces expressions Lambda :

Le code sans les expressions Lambda

```
for(Consumer consumer : consumers) {
    for(Provider provider : providers) {
        if(consumer.getTransactionId() != null && consumer.getTransactionId().equals(provider.getTransactionId())) {
            JsonArchive archive = new JsonArchive(new Date(), consumer, provider);
            archives.add(archive);
            i++;
        }
    }
}
```

Figure 18- Exemple sans expressions Lambda

Le code avec les expressions Lambda et la gestion des exceptions

```
consumers.stream().forEach(consumer -> {
    System.out.println("Consumer : " + consumer.getId());
    providers.stream().forEach(provider -> {
        System.out.println("Provider : " + provider.getId());
        if (consumer.getTransactionId() != null && consumer.getTransactionId().equals(provider.getTransactionId())) {
            JsonArchive archive = null;
            try {
                archive = new JsonArchive(new Date(), encryptConsumerFields(consumer), encryptProviderFields(provider));
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            } catch (GeneralSecurityException e) {
                e.printStackTrace();
            }
            archives.add(archive);
        }
    });
});
```

Figure 19- Exemple avec expressions Lambda

Nous avons utilisé ici les expressions lambda fournies par le java 8, pour parcourir la liste de *consumers* ainsi que celle de *providers* et pour faire le matching entre un *consumer* et son *provider* en se basant sur le « transactionId », ensuite, nous avons effectué le cryptage des deux champs « institute » et « legal_context » avant d’écrire le fichier contenant l’ensemble des objets Archive sur le disque.

4.3.4 Cryptage

Nous avons utilisé les **algorithmes AES+RSA** en java standard. *AES*, un algorithme de cryptage symétrique, ce qui veut dire qu’une seule clé est utilisée pour crypter et décrypter le message. Par contre, *RSA* est un algorithme asymétrique, une paire de clés est utilisée, une privée qu’on garde pour nous et une autre publique qu’on partage avec tout le monde.

Les deux algorithmes sont utilisés à des fins différentes, RSA est utile pour l’échange de clés, mais son utilisation est lente. AES est très rapide, mais souffre des risques de sécurité liés à

l'échange de clés. Un bon compromis consiste à utiliser RSA pour chiffrer la clé symétrique utilisée dans le chiffrement AES des données plus volumineuses.

La figure suivante montre la manière dont nous avons développé le cryptage des archives avec AES+RSA :

```
@Override
public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) throws Exception {
    File file = new File(path);
    File[] listOfFiles = file.listFiles();
    for (int i = 0; i < listOfFiles.length; i++) {
        if (listOfFiles[i].isFile()) {
            byte[] content = FileUtils.readFileToByteArray(listOfFiles[i]);
            byte[] encryptedContent = JavaPGP.encrypt(content, JavaPGP.generateKeyPair().getPublic());
            FileUtils.writeByteArrayToFile(new File(path + listOfFiles[i].getName()),
                encryptedContent);
        }
    }
    return RepeatStatus.FINISHED;
}
```

Figure 20- Cryptage

La méthode *execute* de la *SolrTasklet* permet de charger le fichier généré précédemment par la *Field Encryption Tasklet* et de le sauvegarder sur *Solr* à l'aide de *SolrRepository*.

Nous n'avons crypté que les deux champs qui dévoilent les informations relatives à la transaction comme décrit précédemment, et cela pour la table *consumer* et celle de *provider* :

```
private Provider encryptProviderFields(Provider provider) throws UnsupportedEncodingException, GeneralSecurityException {
    if (provider.getInstitute() != null)
        provider.setInstitute(encryptor.encrypt(provider.getInstitute()));
    if (provider.getLegalContext() != null)
        provider.setLegalContext(encryptor.encrypt(provider.getLegalContext()));
    return provider;
}
```

Figure 21- Cryptage de seulement deux champs

4.3.5 Arborescence des deux applications

- Application d'archivage

```
| .gitignore
| .travis.yml
| pom.xml
| README.md
|
└─src
    └─main
        └─java
            └─com
                └─cirb
                    └─archiver
                        └─ArchiverBatchApplication.java
                        └─batch
                            └─jobs
                                └─ArchivingJob.java
                                └─tasklets
                                    └─FieldsEncryptionTasklet.java
                                    └─SolrTasklet.java
                                    └─utils
                                        └─ArchiveJsonItemAggregator.java
                                        └─JavaPGP.java
                                    └─configuration
                                        └─SolrConfig.java
                                    └─domain
                                        └─Consumer.java
                                        └─JsonArchive.java
                                        └─Provider.java
                                        └─SolrArchive.java
                                    └─repositories
                                        └─ConsumerRepository.java
                                        └─ProviderRepository.java
                                        └─SolrArchiveRepository.java
                    └─resources
                        └─application.yml
```

ArchiverBatchApplication : c'est une classe principale pour démarrer l'application *Spring Boot*.

ArchivingJob: classe de configuration du job responsable de l'archivage.

FieldsEncryptionTasklet: classe de configuration de la tasklet qui a pour rôle d'extraire les données de la base et de crypter les deux champs avant de sauvegarder ces données dans un fichier *JSON*.

SolrTasklet: configuration du *tasklet* qui enregistre le fichier généré par *FieldsEncryptionTasklet* dans *Solr*.

ArchiveJsonItemAggregator: permet de définir comment les objets Java vont être structurés dans le fichier *JSON* dans la *FieldsEncryptionTasklet*.

JavaPGP: contient tous le code métier et algorithme de cryptage/décryptage utilisé lors du chiffage des 2 champs (institute, keyValue) dans *FieldsEncryptionTasklet*.

SolrConfig: contient la configuration du *Solr*, avec des valeurs externalisées dans le fichier « application.yml ».

Et les classes qui restent ce sont des *repositories* où nos méthodes classiques sont définies, et sont parmi les avantages de *Spring Boot*.

- Application de consultation

```

| .gitignore
| archiver-service.iml
| pom.xml
|
└─src
    └─main
        └─java
            └─com
                └─cirb
                    └─archive
                        └─ArchiverServiceApplication.java
                    └─config
                        └─SolrConfig.java
                        └─WebSecurityConfig.java
                    └─security
                        └─JavaPGP.java
                        └─JwtAuthenticationFilter.java
                        └─JwtAuthorizationFilter.java
                        └─SecurityConstants.java
                        └─UserDetailsServiceImpl.java
                    └─domain
                        └─Archive.java
                        └─ArchiveDetails.java
                        └─ArchiveMetadata.java
                        └─JsonArchive.java
                        └─Result.java
                        └─User.java
                    └─vo
                        └─ArchiveVO.java
                        └─SearchVO.java

```

ArchiverServiceApplication: la classe principale qui s'exécute lors de démarrage de l'application.

SolrConfig: contient la configuration de serveur Solr, avec des valeurs externalisées dans le fichier « application.yml ».

WebSecurityConfig: classe de configuration principale du *framework Spring Security*.

JavaPGP: contient tous le code métier et algorithme de décryptage utilisé lors du déchiffrement des 2 champs (institute, keyValue) dans la classe ArchiveService.

JwtAuthenticationFilter: un filtre pour interdire l'accès, à des routes spécifiques, pour les utilisateurs non authentifiés.

JwtAuthorizationFilter: ce filtre permet de vérifier si un utilisateur authentifié a le droit d'accéder à l'application ou pas.

Archive : contient les données qui vont s'afficher dans le tableau

ArchiveDetails: fait la correspondance entre les archives et les données affichées, pour récupérer l'archive à télécharger.

Result: c'est l'objet qui contient tous le détail sur l'opération de la recherche, que ce soit le tableau avec les attributs importants ou les archives à télécharger.


```

| | | |
| | | | └─repositories
| | | |   ArchiveRepository.java
| | | |   UserRepository.java
| | | |
| | | | └─rest
| | | |   SearchController.java
| | | |
| | | | └─service
| | | |   ArchiveService.java
| | | |   IArchiveService.java
| | | |   IUserService.java
| | | |   UserService.java
| | | |
| | | | └─exception
| | | |   NotFoundException.java
| | |
| | └─META-INF
| |   MANIFEST.MF
| |
| └─resources
|   application.yml
|
└─test
  └─java
    └─com
      └─cirb
        └─archive
          └─service
            └─archiverservice
              ArchiverServiceApplicationTests.java

```

SecurityConstants: regroupe l'ensemble des constantes utilisées dans la partie sécurité de l'application.

UserDetailsServiceImpl: contient une méthode 'findByUsername', qui permet de sélectionner un utilisateur à partir de son username.

SearchController: classe qui sert à répondre aux requêtes de la recherche.

ArchiveService: classe qui contient le code métier de recherche des archives dans *Solr*.

NotFoundException: pour déclencher une exception si l'ID de l'archive n'est pas trouvé

ArchiverServiceApplicationTests: contient un test unitaire sur l'API Rest Docs.

IArchiveService, IUserService, UserService:

Contient la logique métier de l'authentification des utilisateurs, recherche et télécharger les archives.

Et les classes qui restent ce sont des *repositories* où nos méthodes classiques sont stockées.

4.3.6 Service RESTful

Nous avons documenté le service RESTful avec l'API Rest Docs, les *snippets* sont automatiquement générés mais dans notre exemple, nous avons rajouté des informations dans ces *snippets* : *request-fields* et *response-body* et *response-fields* :

```
[[resources-index]]
== Archives

The archives resource is used to operate on archives (crud).

[[resources-archives-get]]
=== Get all archives

A `GET` request will retrieve all the stored archives

include::{snippets}/get-all/response-fields.adoc[]

[[resources-archives-search]]
=== Search archives

A `POST` request will retrieve all the archives that matches the criteria.

The request body looks something like this :

include::{snippets}/search/request-fields.adoc[]

And the response you will be receiving is :

include::{snippets}/search/response-body.adoc[]
```

Figure 22- Documentation de service RESTful

Et afin d'assurer le bon fonctionnement, nous avons fait des tests unitaires, prenons l'exemple de la récupération de tous les archives. Par ceci, nous vérifions que la réponse contient ces attributs quand la documentation est générée :

```
@Test
public void getAll() throws Exception {
    this.mockMvc.perform(get("/api/archives").accept(MediaType.APPLICATION_JSON_VALUE))
        .andExpect(status().isOk()).andDo(document("get-all",
            responseFields(
                fieldWithPath("[].id")
                    .description("The archive's id"),
                fieldWithPath("[].content")
                    .description("The content of the archive as byte array"),
                fieldWithPath("[].date")
                    .description("The date of the creation of archive"),
                fieldWithPath("[].extension")
                    .description("Archive extension"),
                fieldWithPath("[].fileName")
                    .description("The archive file name")
            )
        ));
}
```

Figure 23- Test unitaire

4.4 Résultats

4.4.1 Application d'archivage

Cette application nous renvoie le fichier contenant les données qui datent plus qu'un an avec le cryptage de deux champs qui peuvent révéler la transaction s'ils se seront fait connaître.

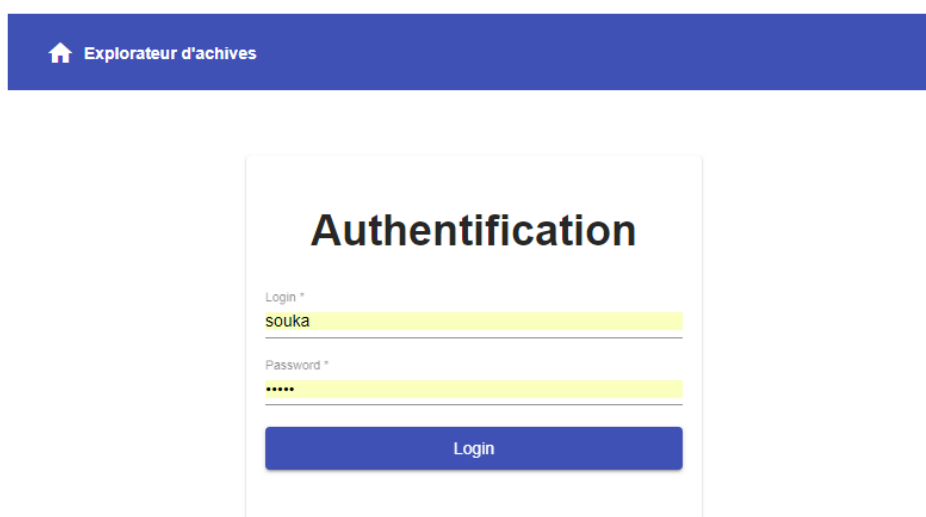
La structure de fichier JSON se fait en se basant sur le numéro de la transaction « transactionId » et puis on renvoie le *provider* et le *consumer* qui ont le même numéro de transaction. Chaque couple « consumer et provider » se caractérise par une *date* avec l'heure dans laquelle l'archivage de la transaction est fait. Comme nous pouvons constater les deux champs « institute » et « legal_context » sont cryptés :

```
{
  "date" : "22/11/2018 21:45:14",
  "consumer" : {
    "id" : 27,
    "action" : "http://fidus.brussels/services/personservice/getIdentity",
    "applicationId" : "fidus-testuser-day",
    "endPoint" : null,
    "error" : "None",
    "internalMessageId" : "a4153d54-be20-4013-88a7-530a69f0c2c2",
    "externalMessageId" : "dba3754d-989f-47b0-86ac-f552ealc50ce",
    "externalMessageContext" : "None",
    "institute" : "T7k+7irdrDwxX5GPvh7wew==",
    "keyType" : "PersonNumber",
    "keyValue" : "93051822361",
    "legalContext" : "Vy6723zwVhajBvx63HO0whZxfwS+G82uaPrNEQNDgqc=",
    "request" : "<soapenv:Body xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:wsu="
    "requestTimestamp" : "2015-09-22 09:56:31.076+02",
    "response" : "OK",
    "responseTimestamp" : "2015-09-22 09:56:31.51+02",
    "transactionId" : "26078baa-8a83-4938-836e-7ceb3a30374b",
    "externalTimestamp" : "2015-09-22"
  },
  "provider" : {
    "id" : 50,
    "action" : "http://kszbcss.fgov.be/PersonIssService/searchPersonBySsin",
    "applicationId" : "PersonService",
    "endPoint" : null,
    "error" : "None",
    "internalMessageId" : "ead387fb-c7b8-4c31-a7b6-1ffbb31ffdl5",
    "externalMessageId" : "11d6f684-9478-9905-865a-458b8ad29e59",
    "externalMessageContext" : "None",
    "institute" : "OR60ldwYaH/nTuAAI03I4w==",
    "keyType" : "PersonNumber",
    "keyValue" : "93051822361",
    "legalContext" : "b4wlFzaYiudU3MQX7A8KpZnvjvmYoxa43sfiz6nnlgQ=",
    "request" : "<soapenv:Body xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:wsu="
    "requestTimestamp" : "2015-09-22 09:56:31.372+02",
    "response" : "OK",
    "responseTimestamp" : "2015-09-22 09:56:31.46+02",
    "transactionId" : "26078baa-8a83-4938-836e-7ceb3a30374b",
    "externalTimestamp" : "2015-09-22"
  }
}
```

Figure 24- Schéma JSON

4.4.2 Application de consultation

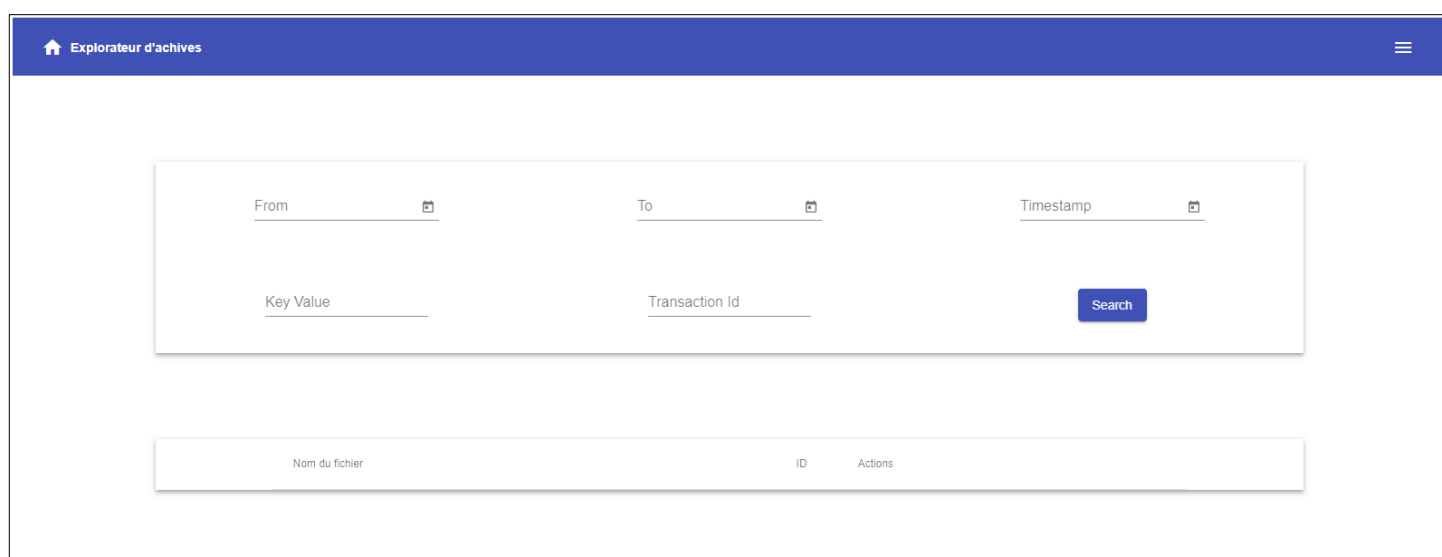
La première interface qui s’affiche est celle de login. Les utilisateurs qui ont droit à consulter les archives ont déjà leur propre compte stocké dans la base de données, ils ne vont pas créer un nouveau pour y accéder donc la possibilité de créer un compte n’est pas nécessaire :



The screenshot shows a login form within a blue header bar labeled 'Explorateur d'archives'. The form is titled 'Authentification' and contains two input fields: 'Login *' with the value 'souka' and 'Password *' with masked characters '.....'. A blue 'Login' button is positioned below the password field.

Figure 25- Interface Login

Une fois l'utilisateur est connecté, il peut faire des recherches soit par rapport au timestamp ou il saisit un intervalle de temps, en entrant le *keyValue* ou encore via un *transaction-id*. Par la suite il obtient tous les archives qui concernent sa demande.





The screenshot shows the search interface. The header bar is blue and labeled 'Explorateur d'archives'. Below the header, there is a search form with five input fields: 'From', 'To', 'Timestamp', 'Key Value', and 'Transaction Id'. Each of the first three fields has a calendar icon to its right. A blue 'Search' button is located to the right of the 'Transaction Id' field. Below the search form, there is a table header with three columns: 'Nom du fichier', 'ID', and 'Actions'.


Figure 26- Interface Recherche

Un exemple de recherche en introduisant un « timestamp » et un « key value » :

archives

From 

To 

Timestamp
12/17/2018 

Key Value
93051822361

Transaction Id

Search

Download archive_16-12-2018.json

Institute	Transaction Id	Legal Context	Key/Value	External Timestamp
CIRB-CIRB	26078baa-8a83-4938-836e-7ceb3a30374b	123234R342	93051822361	sept. 22, 2015
CIRB-CIRB	a6efc571-9730-4bbe-bcbf-8af330dd651b	123234R342	93051822361	sept. 22, 2015
CIRB-CIRB	fe238b30-1701-423d-894b-e53c1cb6c506	123234R342	93051822361	sept. 22, 2015

Figure 27- Exemple de recherche

Et ils peuvent bien évidemment télécharger l'archive ou les archives correspondant à la recherche mais crypté afin de renforcer la sécurité.

4.4.3 Documentation de RESTful

En utilisant ce plugin « asciidoctor-maven-plugin », nous avons produit une documentation précise et bien structurée qui aide principalement notre team à bien comprendre ce que le service RESTful réalise :

[Table of Contents](#)
[Overview](#)
[HTTP verbs](#)
[HTTP status codes](#)
[Resources](#)
[Archives](#)
[Get all archives](#)
[Search archives](#)

Resources

Archives

The archives resource is used to operate on archives (crud).

Get all archives

A `GET` request will retrieve all the stored archives

Path	Type	Description
<code>[] .id</code>	String	The archive's id
<code>[] .content</code>	Array	The content of the archive as byte array
<code>[] .date</code>	String	The date of the creation of archive
<code>[] .extension</code>	String	Archive extension
<code>[] .fileName</code>	String	The archive file name

[Table of Contents](#)
[Overview](#)
[HTTP verbs](#)
[HTTP status codes](#)
[Resources](#)
[Archives](#)
[Get all archives](#)
[Search archives](#)

Search archives

A `POST` request will retrieve all the archives that matches the criteria.

The request body looks something like this :

Path	Type	Description
<code>dateFrom</code>	Null	The minimal date of the archive
<code>dateTo</code>	Null	The max date of the archive
<code>timestamp</code>	Null	The exact date of the archive, used only if neither <code>dateFrom</code> nor <code>dateTo</code> is specified
<code>keyValue</code>	Null	Used to retrieve only archives which <code>keyValue</code> attribute is equals to the specified <code>keyValue</code> attribute
<code>transactionId</code>	Null	Used to retrieve only archives which <code>transactionId</code> attribute is equals to the specified <code>keyValue</code> <code>transactionId</code>

And the response you will be receiving is :

```
{"details":[{"id":"1263219d-e62c-48d2-a955-f2c410066494","fileName":"archive_16-12-2018.json"}],"archives"}
```

Figure 28- Documentation RESTful

V. Conclusion et perspectives

L'objectif de ce Travail de Fin d'Études consistait à réaliser l'archivage d'un *audit log* contenant l'ensemble d'échange d'information privée entre les institutions publiques Bruxelloises, et de pouvoir consulter ces archives tout en veillant à ce que les données ne soient pas dévoilées.

Pour ce faire, nous avons effectué une analyse fonctionnelle et technique de cette problématique. Ceci qui nous a permis de faire un choix vis-à-vis les technologies à utiliser et de développer les différentes applications.

Dans un premier temps, il a fallu avoir une idée complète sur ce que l'infrastructure Fidus a pu apporter pour la région Bruxelloise. Le but était de comprendre la source de la problématique.

En plus de ce que le chef de projet a partagé lors de la première journée en stage, la plateforme confluence de CIRB nous a énormément aidés en nous permettant de consulter des diagrammes UML qui expliquaient les différentes fonctionnalités de la plateforme Fidus.

Après avoir fait l'analyse fonctionnelle et technique, nous avons désigné les technologies qui mènent à bien ce projet. Nous avons commencé à développer un *Batch* qui s'exécute chaque année à travers une application *Spring Batch* avec *Spring Boot* qui permet d'extraire et d'archiver les données qui datent de plus d'un an sous format *JSON*. Puis, nous avons crypté ces archives par le biais d'algorithmes de cryptage. Il est important de préciser que la sécurité a toujours été l'enjeu le plus crucial de cette infrastructure solution.

Ensuite, nous avons développé une deuxième application *Spring Boot* de consultation qui n'était pas dans le *scope* mais que nous avons pensé de la mettre en défi. Le but était de toucher d'autres technologies spécialement *Angular6*, et donc une troisième application *front-end* s'est rajoutée.

Et pour finir, nous pouvons affirmer que notre solution a pu répondre aux attentes du chef de projet ainsi que l'architecte. Mais, elle reste toujours un prototype qui nécessite sûrement des améliorations dans le futur.

En perspective, le travail que nous avons réalisé pourrait être amélioré par l'introduction de nouveaux aspects à savoir :

- Déléguer le système d'authentification à un serveur d'authentification, par exemple *Keycloak* qui est un logiciel open source permettant l'authentification unique avec la gestion des identités et des accès, ce qui sécurise au mieux les archives comme la sécurité est l'élément indispensable au sein du CIRB.
- Mais aussi, nous pourrions rendre le déploiement encore plus facile en travaillant avec *Docker* qui est à son tour un logiciel libre permettant de lancer d'une manière facile des applications dans des conteneurs logiciels.

VI. Bibliographie

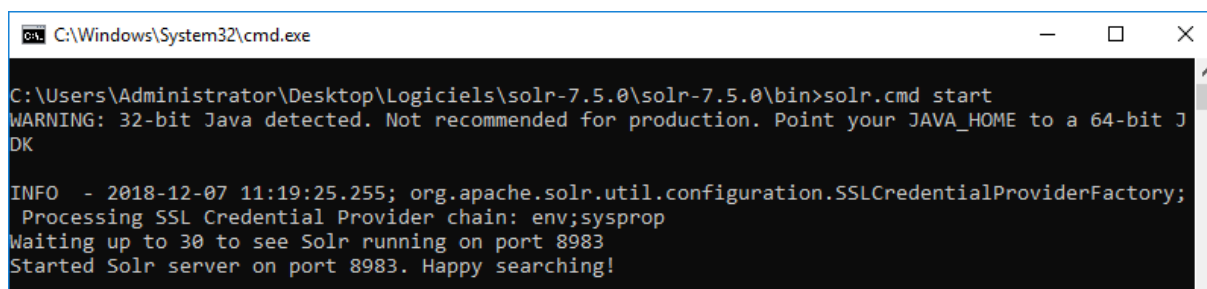
Toutes les sources ci-dessous ont été consultées entre 17 août et 19 décembre 2018.

- [1] CIRB web site, https://bric.brussels/en?set_language=en.
- [2] Fidus platform, https://bric.brussels/en/our-solutions/infrastructure-solutions/fidus?set_language=en.
- [3] Spring Batch Documentation, <https://docs.spring.io/spring-batch/4.0.x/reference/html/index.html>
- [4] Spring Boot Documentation, <https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/html/>
- [5] Spring Data JPA Documentation, <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>
- [6] Spring Security Documentation, Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll and others, <https://docs.spring.io/springsecurity/site/docs/3.0.x/reference/springsecurity.html>
- [7] Implementing JWT with Spring Boot and Spring Security, <https://medium.com/@xoor/jwt-authentication-service-44658409e12c>
- [8] CRUD Application Using Spring Data Solr and Spring Boot, <https://dzone.com/articles/crud-application-using-spring-data-solr-and-spring>
- [9] Spring Batch with Spring Boot, <https://www.opencodez.com/java/spring-batch-with-spring-boot.htm>
- [10] Spring Batch + Spring Boot Java Config Example, Lokesh Gupta, <https://howtodoinjava.com/spring-batch/java-config-multiple-steps/>
- [11] Solr server, <http://lucene.apache.org/solr/>
- [12] Solr Tutorial, open source search and analytics, <http://yonik.com/solr-tutorial/>
- [13] Spring data Solr, <https://www.baeldung.com/spring-data-solr>
- [14] Solr, <http://yonik.com/solr-tutorial/>
- [15] Introduction to Spring Data Solr, <https://www.baeldung.com/spring-data-solr>
- [16] Spring Data for Apache Solr, <https://spring.io/projects/spring-data-solr>
- [17] Apache Solr Tutorial for Beginners, Veeramani Kalyanasundaram <https://examples.javacodegeeks.com/enterprise-java/apache%20solr/apache-solr-tutorial-beginners/>
- [18] Solr Ref Guide, https://lucene.apache.org/solr/guide/7_6/solr-tutorial.html
- [19] How does RSA and AES differ, Hrishabkumar Jha, <https://www.quora.com/How-does-RSA-and-AES-differ>

- [20] Java Cryptography, Jakob Jenkov, <http://tutorials.jenkov.com/java-cryptography/index.html>
- [21] Java Cipher, Jakob Jenkov, <http://tutorials.jenkov.com/java-cryptography/cipher.html>
- [22] AES Encryption and Decryption in Java, Dhiraj Ray, <https://www.javacodegeeks.com/2018/03/aes-encryption-and-decryption-in-javacbc-mode.html>
- [23] Encrypting and Decrypting Files in Java, Baeldung, <https://www.baeldung.com/java-cipher-input-output-stream>
- [24] How to encrypt String in Java, <https://stackoverflow.com/questions/1205135/how-to-encrypt-string-in-java>
- [25] PGP Encryption and Decryption with Java, <https://stackoverflow.com/questions/9596298/pgp-encryption-and-decryption-with-java>
- [26] Java 8 et expressions lambda, <https://www.baeldung.com/java8>
- [27] Java 8 Lambda Basics, Koushik Kothagal, https://javabrainz.io/courses/java_lambdabasics/
- [28] Expressions lambda, <https://www.supinfo.com/articles/>
- [29] Microservices avec Spring Boot, <https://medium.com/omarelgabrys-blog/microservices-with-spring-boot-intro-to-microservices-part-1-c0d24cd422c3>
- [30] Créez un Microservice grâce à Spring Boot, <https://openclassrooms.com/fr/courses/4668056-construisez-des-microservices/5122884-creez-un-microservice-grace-a-spring-boot>
<https://examples.javacodegeeks.com/enterprise-java/apache-solr/apache-solr-tutorial-beginners/>
- [31] Angular tutorials, <https://angular.io/tutorial>
- [32] Angular 7 Tutorial, Learn Angular by Example, Gary Simon, <https://coursetro.com/posts/code/171/Angular-7-Tutorial---Learn-Angular-7-by-Example>
- [33] Spring REST Docs, <https://spring.io/projects/spring-restdocs>

VII. Manuel d'utilisation

Après avoir téléchargé *Solr*, on le démarre comme suite:



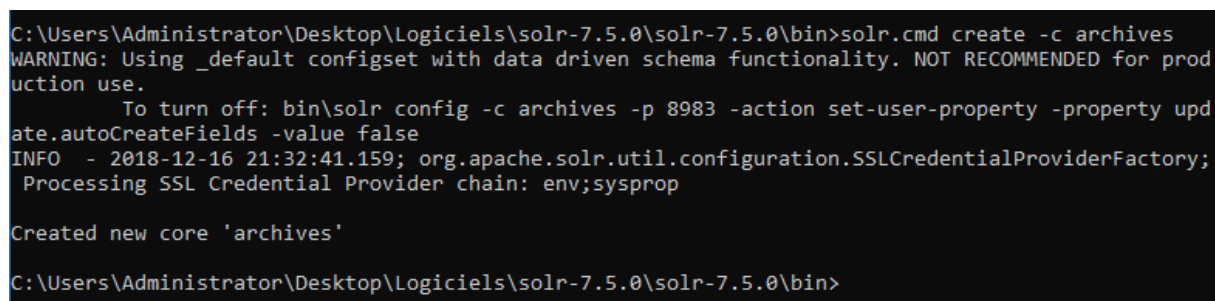
```
C:\Windows\System32\cmd.exe

C:\Users\Administrator\Desktop\Logiciels\solr-7.5.0\solr-7.5.0\bin>solr.cmd start
WARNING: 32-bit Java detected. Not recommended for production. Point your JAVA_HOME to a 64-bit J
DK

INFO - 2018-12-07 11:19:25.255; org.apache.solr.util.configuration.SSLCredentialProviderFactory;
Processing SSL Credential Provider chain: env;sysprop
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!
```

Figure 29- Démarrage de serveur Solr

On créera un *core* « Archives » :



```
C:\Users\Administrator\Desktop\Logiciels\solr-7.5.0\solr-7.5.0\bin>solr.cmd create -c archives
WARNING: Using _default configset with data driven schema functionality. NOT RECOMMENDED for prod
uction use.
To turn off: bin\solr config -c archives -p 8983 -action set-user-property -property updat
ate.autoCreateFields -value false
INFO - 2018-12-16 21:32:41.159; org.apache.solr.util.configuration.SSLCredentialProviderFactory;
Processing SSL Credential Provider chain: env;sysprop

Created new core 'archives'

C:\Users\Administrator\Desktop\Logiciels\solr-7.5.0\solr-7.5.0\bin>
```

Figure 30- Création d'un core

Ensuite, on accède au projet *Angular6* afin d'installer les packages en tapant la commande *npm install* dans l'invite de commande:

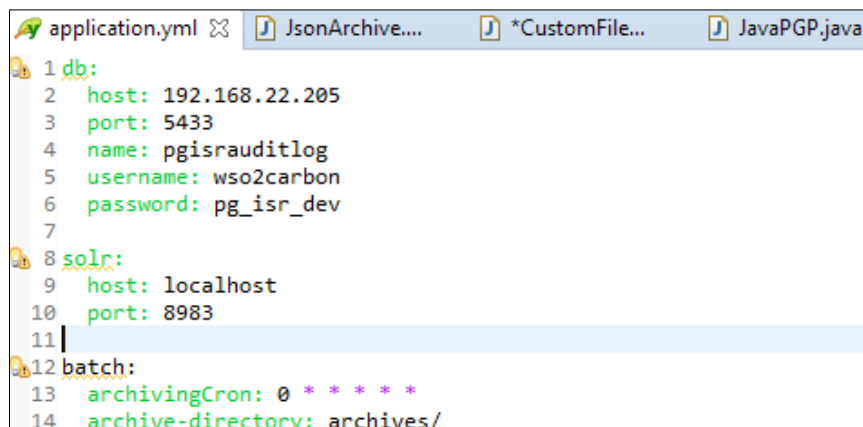


```
C:\Users\Administrator\Desktop\archiver-ui-master\archiver-ui-master>npm install
[.....] / extract:camelcase: http fetch GET 200 https://registry.npmjs.org/cacache 613ms
```

Figure 31- Commande installation package

Et puis pour les bases de données, on a une sur le serveur nommée « *pgisrauditlog* » et une autre que nous avons créé localement pour les utilisateurs qui ont droit à consulter les archives « *archiveservice* ».

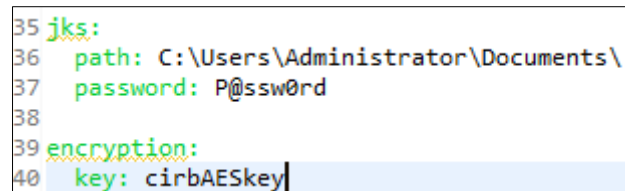
Et vu que *Spring boot* met à notre disposition un fichier `application.yml` qui nous permet de configurer nos applications, donc toute configuration est répertoriée dans ce fichier :



```
1 db:
2   host: 192.168.22.205
3   port: 5433
4   name: pgisrauditlog
5   username: wso2carbon
6   password: pg_isr_dev
7
8 solr:
9   host: localhost
10  port: 8983
11
12 batch:
13   archivingCron: 0 * * * * *
14   archive-directory: archives/
```

Figure 32- Fichier de configuration archivage

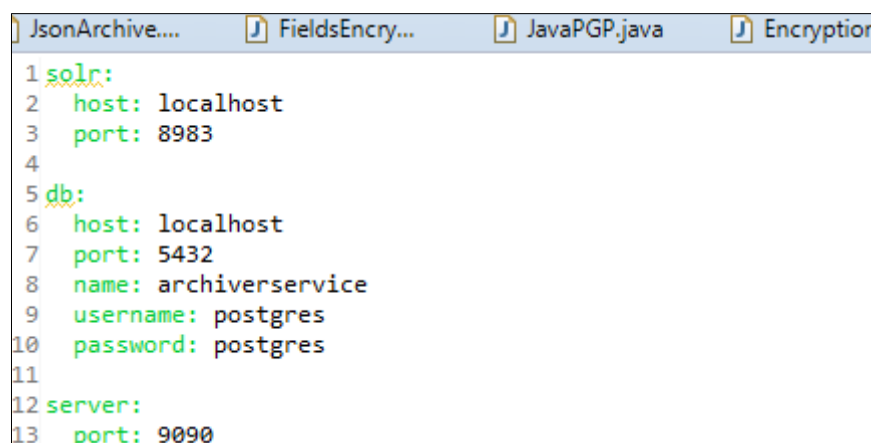
Et à signaler qu'il faut changer le chemin « path » où la clé de cryptage se stocke :



```
35 jks:
36   path: C:\Users\Administrator\Documents\
37   password: P@ssw0rd
38
39 encryption:
40   key: cirbAESkey
```

Figure 33- Chemin de la clé de cryptage

La configuration de la deuxième application où la seule différence est le rajout de port de serveur qui est le 9090 :



```
1 solr:
2   host: localhost
3   port: 8983
4
5 db:
6   host: localhost
7   port: 5432
8   name: archiverservice
9   username: postgres
10  password: postgres
11
12 server:
13   port: 9090
```

Figure 34- Fichier de configuration consultation

Le fichier `application.yml` est bien structuré et sa première utilité est de faciliter certaines configurations ainsi que la modification de ces dernières.

VIII. Annexes

9.1 Méthodes Agiles et Scrum

Les méthodes agiles sont des méthodes dédiées à la gestion de projet. Elles se basent sur un processus de développement itératif et incrémental qui s'adapte aux changements des besoins du client. Une méthode Agile génère un produit de haute qualité tout en prenant en compte l'évolution des besoins des clients.

Scrum est une méthode Agile qui permet de produire la plus grande valeur métier dans la durée la plus courte. Avec Scrum, la réalisation d'un ensemble de fonctionnalités est faite par itération d'une durée fixe « *sprint* » qui est entre 1 à 4 semaines, ce qui assure la livraison d'un produit partiel fonctionnel par itération.



Figure 35- Processus Scrum

Scrum définit 3 rôles :

Le product owner : définit les fonctionnalités du produit et les valide mais aussi il définit les priorités dans le *backlog* en fonction de la valeur « métier ». Généralement, il représente le client.

Team Scrum : c'est l'équipe de développement multidisciplinaire qui réalise le produit et décide elle-même comment travailler ensemble, et ne doit pas changer pendant un Sprint.

Scrum master : assure que l'équipe est complètement fonctionnelle et productive, protège l'équipe des interférences extérieures, élimine les obstacles et s'assurer que les principes de Scrum sont respectés.

Le cycle de vie d'un projet *Scrum* est rythmé par une série de réunion :

Planification du Sprint : lors de cette réunion le Product Owner expose le contenu du Product Backlog et définit les tâches prioritaires. L'équipe de développement choisit à partir du backlog de produit les éléments qu'elle estime pouvoir réaliser au cours du prochain sprint et évalue la charge du travail de chaque PEB,

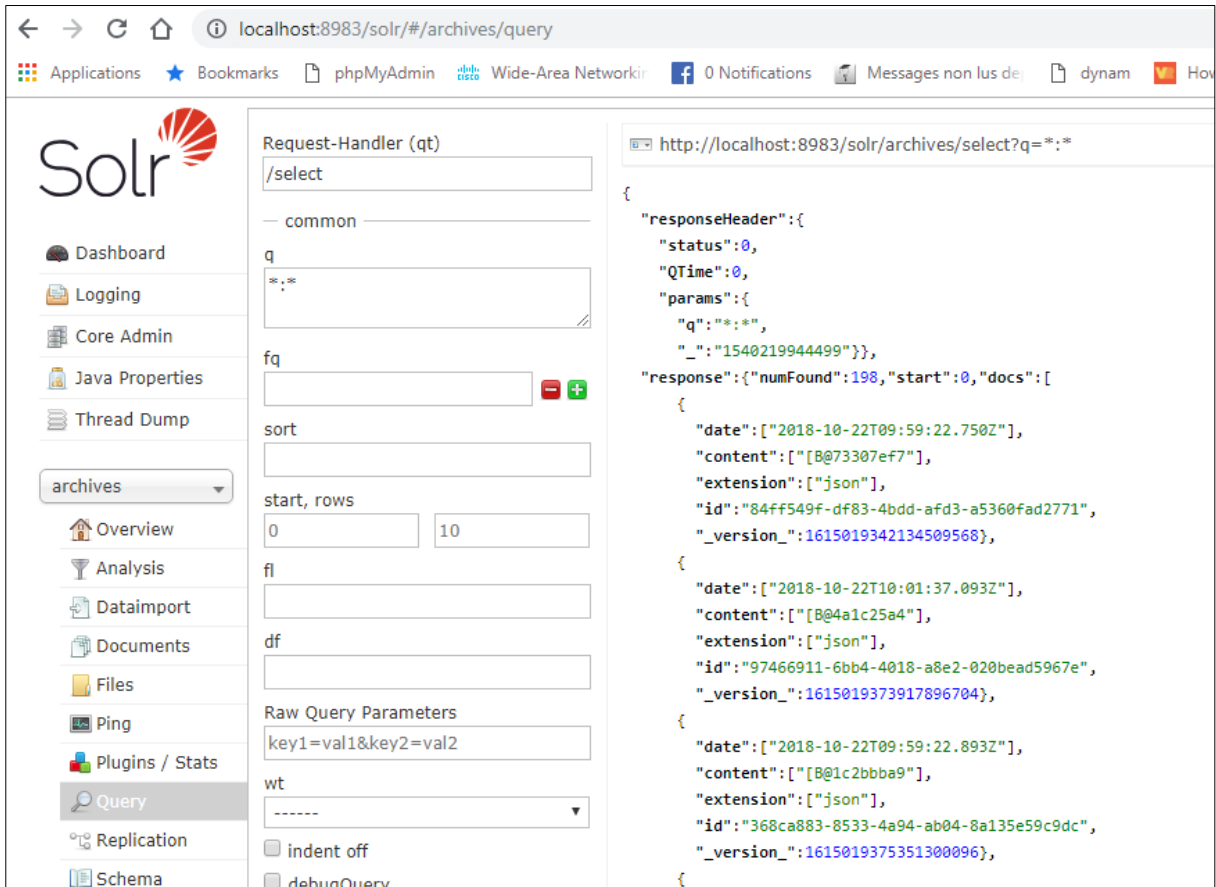
Mêlée quotidienne : réunion de 15 minutes maximum qui se fait debout pendant laquelle chacun répond à trois questions : « qu'est-ce que j'ai terminé depuis la dernière mêlée », « qu'est-ce que je compte faire d'ici la prochaine mêlée » et quels obstacles me retardent ?

Revue de sprint : réunion qui a lieu à la fin du sprint et au cours de laquelle l'équipe de développement fait une démonstration de « sous-produit » réalisé. Le *Product Owner* valide les PBI complets et remplace ceux non finis dans le *product backlog* et c'est lui qui décide si le projet est terminé ou pas. Le *Scrum Master* oriente la rencontre pour ne pas dépasser le temps.

Rétrospective de sprint: afin d'améliorer les performances pour les nouvelles tâches. L'équipe de développement et le *Scrum Master* fait une analyse rétrospective du sprint afin d'optimiser les sprints suivants. Le rôle du SM est alors de garder une conversation constructive.

9.2 Indexation des archives sous Solr

La classe archive contient, l'id se génère quand on fait le sauvegarde de Repository afin de stocker l'objet SolrArchive, le contenu de fichier en byte et quand on veut le télécharger il fait le mapping entre byte et le contenu du fichier, l'extension et la date actuelle dans laquelle l'archive a été créé.



```
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "q": "*:*",
      "_: "1540219944499"}},
  "response": {
    "numFound": 198,
    "start": 0,
    "docs": [
      {
        "date": ["2018-10-22T09:59:22.750Z"],
        "content": ["[B@73307ef7"],
        "extension": ["json"],
        "id": "84ff549f-df83-4bdd-afd3-a5360fad2771",
        "_version_": 1615019342134509568},
      {
        "date": ["2018-10-22T10:01:37.093Z"],
        "content": ["[B@4a1c25a4"],
        "extension": ["json"],
        "id": "97466911-6bb4-4018-a8e2-020bead5967e",
        "_version_": 1615019373917896704},
      {
        "date": ["2018-10-22T09:59:22.893Z"],
        "content": ["[B@1c2bbba9"],
        "extension": ["json"],
        "id": "368ca883-8533-4a94-ab04-8a135e59c9dc",
        "_version_": 1615019375351300096},
      {
```

Figure 36- Stockage sous Solr

Le nom du fichier contient la date de l'archivage, ce qui distingue les différents fichiers archivés mais encore facilite la consultation de ces derniers.

IX. Glossaire

Intégrateur de Services : une institution qui, par ou en vertu d'un traité, d'un règlement, d'une directive, d'une loi, d'un décret ou d'une ordonnance, est chargée de l'intégration de services à un niveau de pouvoir ou dans un secteur déterminé.

Donnée Authentique : donnée récoltée et gérée par un service public participant à une base de données et qui fait foi comme donnée unique et originale concernant la personne ou le fait de droit concerné, de sorte que d'autres instances ne doivent plus collecter cette même donnée.

Source Authentique : banque de données dans laquelle sont conservées des données authentiques.

ESB : **E**ntreprise **S**ervice **B**us, qui est une technique informatique permettant la communication des applications qui n'ont pas été conçues pour fonctionner ensemble.

CISO : **C**hief **I**nformation **S**ecurity **O**fficer, est l'expert qui garantit la sécurité, la disponibilité et l'intégrité du système d'information et des données.

RESTful : adjectif qualifiant une architecture de type REST.

Front-end : couche de présentation ou partie publique de l'application. Elle correspond à l'interface client de l'application.

Back-end : couche d'accès aux données ou partie réservée aux administrateurs de l'application. Elle correspond au serveur et à ce qui lui est lié.

JSON : **J**ava**S**cript **O**bject **N**otation est un format léger d'échange de données dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée. Il est facile à lire ou à écrire pour des humains et aisément analysable par des machines.