Université Paris 8 - Vincennes à Saint-Denis

**Licence informatique**

**Rapport de stage**

# Traducteur de données

## b2b-Endis et b2b-Engie

**Habib BALIT**

Numéro étudiant: 17 80 19 13

Organisme d'accueil : Energisme
Tuteur – Organisme d'accueil: Farid Faoudi
Tuteur – Université: Jean-Noël Vittaut

2018/2019

# Contents

**1 Introduction générale** **2**

**2 Développement** **3**
2.1 Installation . . . . . 3
2.2 Training . . . . . 3
2.3 b2b-Enedis (Java) . . . . . 4
2.3.1 Introduction . . . . . 4
2.3.2 Implémentation . . . . . 4
2.3.3 Conclusion . . . . . 8
2.4 b2b-Engie (Python) . . . . . 9
2.4.1 Introduction . . . . . 9
2.4.2 Implémentation . . . . . 9
2.4.3 Conclusion . . . . . 12
2.5 b2b-meter (Java) . . . . . 13
2.5.1 Introduction . . . . . 13
2.5.2 Implémentation . . . . . 14
2.5.3 Conclusion . . . . . 14

**3 Conclusion générale** **16**

# 1 Introduction générale

Dans le cadre du stage de fin de licence informatique, j'ai effectué ce dernier du 11 juin 2019 aux 09 aoûte 2019 (soit l'équivalent de deux mois), au sein de l'entreprise Energisme.
L'entreprise Energisme située à Boulogne-Billancourt est la plateforme Big Data et IoT à destination des acteurs de la performance énergétique. Conçue autour de technologies innovantes, elle permet de collecter et d'agréger tous types de données multi-fluides provenant de sources hétérogènes. Ces données sont ensuite fiabilisées et restituées via des tableaux de bords multi-niveaux et multi-rôles. Energisme met à la disposition de ses clients des outils de modélisation prédictive ainsi qu'un écosystème de partenaires pour anticiper et optimiser leurs consommations. Mon maître de stage étant Farid Faoudi, j'ai pu apprendre dans d'excellentes conditions et j'ai bénéficié d'un soutien de qualité pendant mon stage.

Mon stage au sein de l'équipe back-end d'Energisme, consisté essentiellement à l'implémentation de traducteur de données, afin de préparer et de traiter ces données reçus par différents clients vers un format standard utilisé par l'entreprise.

# 2 Développement

## 2.1 Installation

A mon arrivé à l'entreprise, une visite guidée était primordial afin de me familiariser avec le milieu du travail, de faire connaissance avec les futures collègues également. Après cela, on passe à notre poste de travail pour faire le point sur tous les outils mis à ma disposition (PC, 2 écrans, souris, clavier, etc.). Ainsi, une installation de l'environnement de travail et obligatoire, car la machine reçue ne contient qu'un système d'exploitation Windows (qu'on peut rien faire avec).
Donc j'ai réalisé les tâches suivantes :

- Installation d'Ubuntu 18.04.
- Installation d'OpenJDK, Python, PIP, GCC, etc.
- Installation d'Intellij idea (éditeur java).
- Installation de Pycharm (éditeur Python).
- Installation de Maven.
- Installation de Postman.
- Configuration des variable global JAVA_HOME, etc.
- Installation et configuration du client Barracuda VPN.

## 2.2 Training

Avant de m'initier vraiment dans le vif du sujet de mon stage, j'ai commencé par faire des petits exercices dans le but de m'adapter et de me familiariser avec la méthode de travail de l'équipe back-end. Ces exercices avaient pour but de m'aider à me familiariser avec la manière du développement de l'équipe, parmi ces exercices :

- GitLab :
  - "Fork" le projet.
  - "Clone" le projet.
  - Travailler sur une branche "develop".
  - Faire des modifications on faisant des commits régulièrement, pour garder des traces.
  - Squash les commits avec leurs façon.
  - Poussé le code à la fin.
  - "Merge request" pour un développeur senior.
- Java :
  - Création et configuration d'un projet maven.
  - Gérer les plugins et les dépendances nécessaire.

- Coder des exercices sur la notion de "marshalling" et "unmarshalling".
- Travailler sur une première facture au format JSON(JSON → java et java → JSON).

- Python :
  - Les modules, les packages, les classes,
  - La notion de "marshalling" et "unmarshalling".

## 2.3 b2b-Enedis (Java)

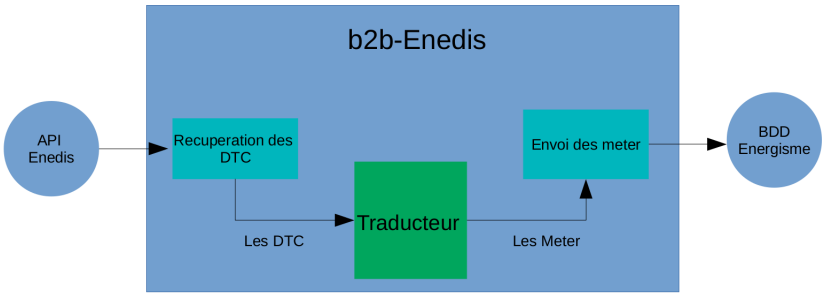


Figure 1: schéma explicatif de b2b-Enedis

### 2.3.1 Introduction

Ce projet consiste à implémenter un traducteur de données techniques et contractuelles (DTC) d'Enedis vers des "meter" Energisme. Le projet est né d'un besoin de récupérer des données qui figurent sur ces DTC, afin de préparer ces données et de réaliser des courbes ou des graphes des puissances souscrites. Les détails nécessaires afin de réaliser le traducteur figure sur un ticket JIRA, qui exprime le besoin, le résultat attendu et le mapping qu'il faut faire. De façon plus semple :

- Les données : des DTC (données technique et contractuelle) fournies par Enedis au format XML dans leur API.
- Le traducteur : notre traducteur va récupérer les informations nécessaires des fichiers XML.
- Le résultat : les informations récupérées seront mises dans des objet "meter" après dans des fichiers JSON, selon un format de l'entreprise.

Ma tâche consisté à travailler sur le carré vert "le traducteur". Alors j'ai procédé de la manière qu'on voit sur la *figure 2*.

### 2.3.2 Implémentation

Ce projet contient 4 packages essentiels(unmarshalling, meter, mapping et types) et une classe principale main, la *figure 2* représente à schéma explicatif, voilà l'utilité de chaque composante :

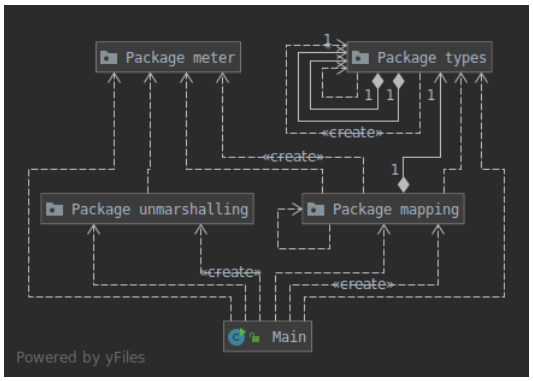


Figure 2: Architecture du traducteur

**Classe main :** c'est la classe principale qui prend les données d'entrée (les DTC) sous forme de répertoire qui les contient. Notre main reçoit deux arguments, le dossier source (qui contient les DTC au format XML) et le dossier de destination (qui va contenir les meters au format JSON).

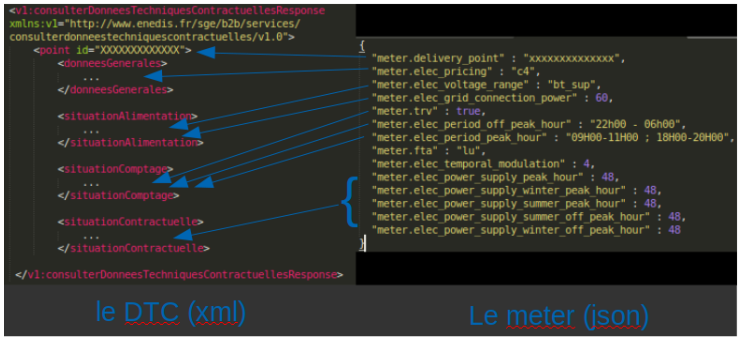


Figure 3: DTC vs Meter

Elle contient aussi une méthode "translateDTC" qui prend les deux arguments si dessus et réalise les opérations suivantes sur tous les DTC :

- "unmarshaling" (fichier XML → objet JAVA).
- traduction de l'objet java avec le bon mapping vers un autre objet java (objet JAVA → objet JAVA).
- "marshalling" (objet java → fichier JSON).

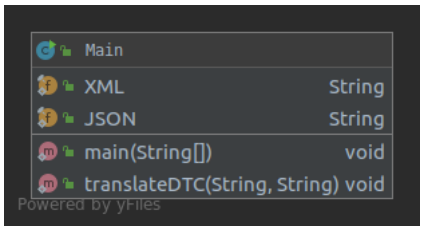


Figure 4: Diagramme de classe de Main

**Package unmarshalling :** ce package contient deux classes qui nous permet de faire :

- fichier XML → objet JAVA
- objet JAVA → fichier JSON

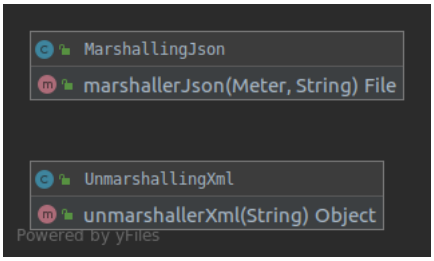


Figure 5: diagramme de classe explicatif du package unmarshalling

**Package types :** ce package contient toutes les classes généré grâce à un outil qui s'appelle "xjc" de JAXB, cette commande permet de générer un jeu de classes à l'aide d'un schéma xsd. Le schéma utilisé afin de générer ces classes est composé de plusieurs documents XSD fourni par Enedis, ces documents permettant de définir la structure et le type de contenu de leurs documents XML (les DTC).
Pour l'utilisation de cette commande, il suffit de taper en ligne de commande : xjc -p nomDuPackage schema.xsd
Grâce à ces classes générées, on peut faire de l'unmarshalling, c'est-à-dire avoir un objet java pour un fichier XML.

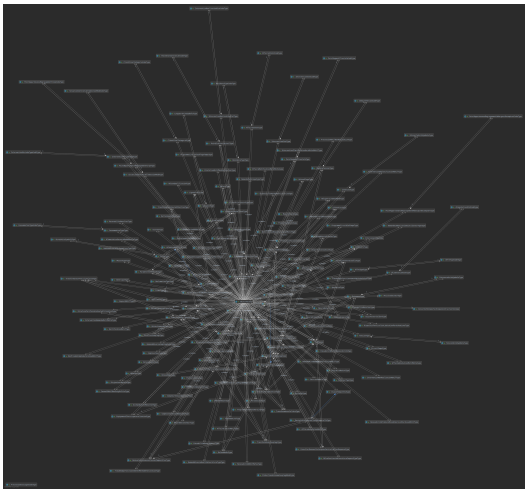

Figure 6: diagramme de classe explicatif du package types, 187 classes générer

**Package meter :** contient la classe Meter.java, qui représente l'objet java qui va devenir le fichier JSON. Cette classe contient tous les noms de clés que peut avoir le fichier JSON et une Map qui va contenir des clés et leurs valeurs d'une instance de cette classe.

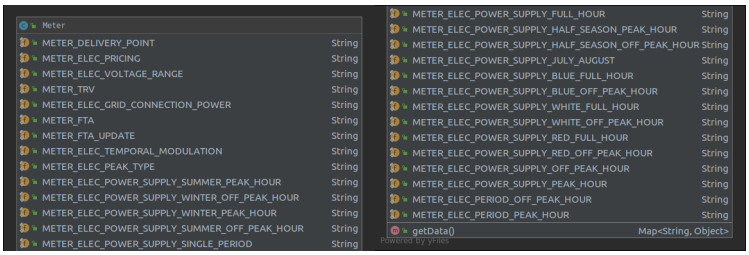


Figure 7: diagramme de classe de Meter.java

**Package mapping :** ce package contient toutes les classes nécessaire à la traduction et au mapping qu'il faut réaliser.

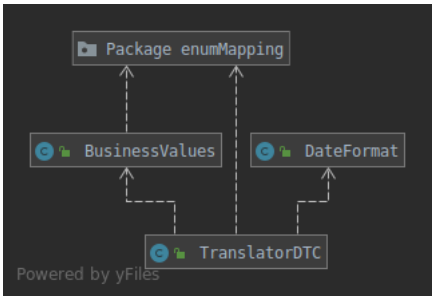


Figure 8: diagramme de classe explicatif du package mapping

Il contient 3 classes principales :

- TranslatorDTC :

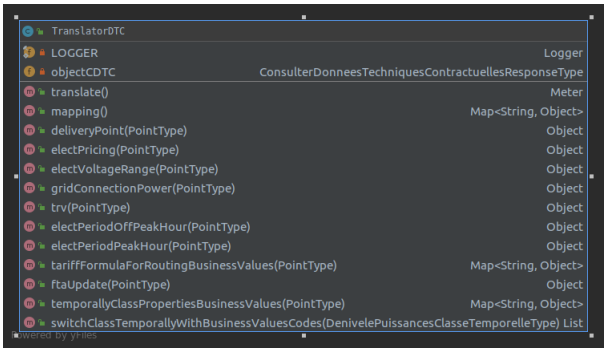


Figure 9: diagramme de classe de TranslatorDTC.java

Cette classe, elle s'occupe de la traduction, ce qui veut dire récupérer de telles valeurs d'un objet java (ConsultationDonneesTechniquesContractuellesResponseType) et les placer dans de tels champs dans un autre objet java (meter), on respecte certain règles.

Il y a deux méthodes principales "translate" et "mapping" ainsi que d'autres méthodes :

– "mapping" : renvoi une Map qui contient tous les champs à récupérer dans l'objet qui représente le fichier XML (CDTC), pour chaque clé

de la Map on lui associer la valeur trouvée dans le fichier XML sinon on lui associer un null.

  - "translate" : renvoi un objet Meter initialisé avec la Map que renvoi "mapping" en filtrant tous les valeurs null, donc on garde une Map qui contient les valeurs trouvent dans le (CDTC).
  - les autres méthodes chacune récupérer une valeur dans l'objet (CDTC) associer à un champ qui porte presque le même nom que la méthode, par ailleurs d'autres méthodes récupèrent une Map de clé et valeur, car elle récupère un ensemble de valeur.

- BusinessValues : est un singleton qui contient une map, qui nous permet de faire le mapping sur différentes valeurs rencontrai, avec d'autre valeurs qu'on a décidé au préalable (BTINF → bt_inf, C1 → c1, etc., où C1, BTNF sont des énumérations).
- DateFormat : est une classe static qui nous permet de convertir des périodes vers un format standard selon la norme iso 8601 :
  - 09H00-11H00;18H00-20H00 → T09 :00/T11:00;T18:00/T20:00
  - 22h00 - 06h00 → T22:00/T06:00
  - etc.

Il contient aussi un package "enumMapping", que lui contient toutes les énumérations possibles.

### 2.3.3 Conclusion

Durant l'implémentation de b2b-enedis j'ai pu utiliser toutes mes connaissances déjà accise pendant ma licence:

- Versioning GitLab et git.
- Réalisation d'application en java : tous qu'est convention de nommage, modularité de code et optimisation de code.
- Documentation de code.

Cela m'a permis d'aiguiser et d'enrichir mes connaissance, et j'ai pu aussi utiliser des concepts que je n'ai pas eu à utiliser auparavant, notamment :

- Réalisation d'application java à l'aide de maven.
- L'utilisation de plusieurs API : JAXB, JAXP, Jackson et Gson.
- L'utilisation de test unitaire à laide de juint et assertj.
- L'utilisation d'un système de "logging" afin de déboguer et suivre les erreurs à laide de slf4j.Logger.
- L'utilisation des Streams, des lambdas et des références de méthodes qui m'a permis d'avoir un code plus facile à comprendre et plus maintenable.

Cette première expérience m'a poussé à travailler davantage, et d'avoir plus de confiance en soi, car ce qu'on produit est utilisé par l'entreprise, ce qui rend ce projet utile.

## 2.4 b2b-Engie (Python)

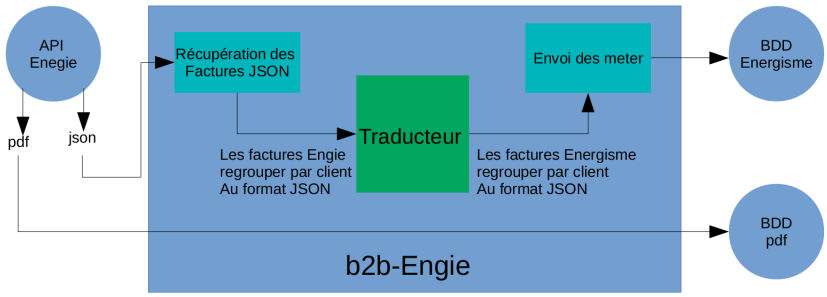


Figure 10: schéma explicatif du b2b-engie

### 2.4.1 Introduction

Ce projet consiste à implémenter un traducteur des factures Engie vers des factures Energisme. Les détails nécessaires afin de réaliser le traducteur, figurent sur un ticket JIRA, qui exprime le besoin, le résultat attendu et le mapping qu'il faut faire. En gros :

- Les données : les factures Engie au format JSON fourni par l'API Engie, les données fournies à notre traducteur et un répertoire qui contient plusieurs répertoires (un répertoire par partenaire). Chaque répertoire contient deux fichiers "getFactureDetails.json" et "getFactures.json", nous utilisons le "getFactureDetails.json" qui contient des milliers de factures, chaque ligne correspondant à une liste de factures pour un client.

```
{"liste":[{facture_1} {facture_2} ... {facture_m}]} client_1
{"liste":[{facture_1} {facture_2} ... {facture_m}]} client_2
...
{"liste":[{facture_1} {facture_2} ... {facture_m}]} client_n
```

Figure 11: Exemple de fichier pour un partenaire

- le traducteur : notre traducteur qui va récupérer des informations de la facture Engie au format JSON et les placer dans la facture Energisme au format JSON.
- le résultat : les factures Energisme au format JSON.

Ma tâche consisté à travailler sur le carré vert "le traducteur". Alors j'ai procédé de la manière suivante.

### 2.4.2 Implémentation

Ce projet est constitué de deux packages :

**package translator :** ce module constitue le traducteur lui-même, qui est à son tour regroupe différents modules (*figure 12*)

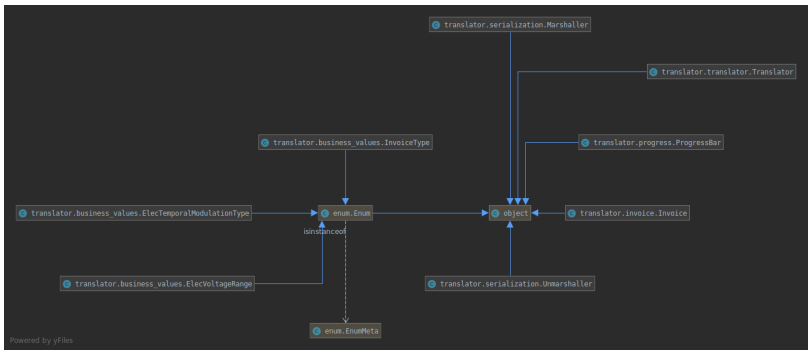


Figure 12: Architecture du package translator

- **__main__ :** c'est le module qui contient le programme principal qui gère les arguments d'entrée, la fonction d'usage et la fonction qui s'occupe de lire les fichiers d'entrées, appelle la classe qui fait la traduction et sauvegarder le résultat dans le répertoire de sorite.
  La fonction principale de ce module prend deux arguments, le répertoire source (contiens plusieurs répertoires, un par partenaire) et le répertoire de destination (un fichier par partenaire), réalise :

  - Lire un par un et ligne par ligne le fichier "getFactureDetails.json" de chaque partenaire.
  - Chaque ligne est une facture, alors on fait de l'unmarshalling fichier (JSON → objet Python (dict)).
  - On traduit l'objet python ci-dessus avec notre classe tranlator qui renvoie un autre objet (objet Python → objet Python(Invoice)).
  - On sauvegarde tous les objets traduits du même fichier "getFacture-Details.json" dans un autre fichier qui porte comme nom "le nom du partenaire exemple : perial-normalized_invoice.json" dans le dossier de destination.

- **sérialisation :** ce module contient deux classes utilitaires avec plusieurs méthodes qui nous permet de sérialiser et dé-sérialiser: Avec des méthodes qui acceptent de lire un fichier ou un String, d'écrire sans indentation ou avec indentation.

  - "unmarshalling" : fichier JSON → objet Python.
  - "marshalling" : objet Python → fichier JSON.

- **invoice :** ce module contient la classe qui représente une facture Energisme avec son format simple et les noms de clé en anglais. Cette classe contient tous les noms de clés possibles pour une facture et contient un dictionnaire qui contient la Map de clé valeur d'une instance de cette classe.

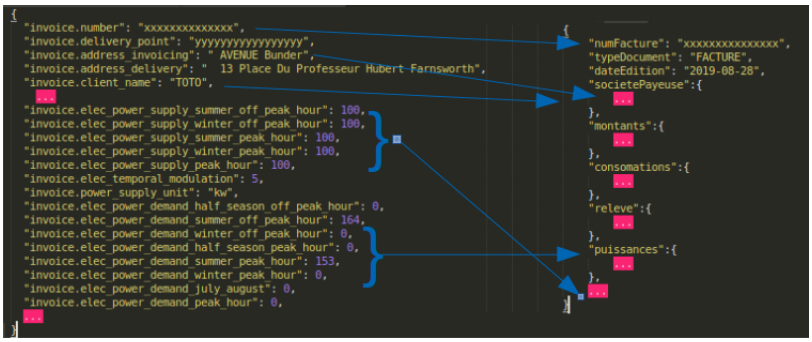


Figure 13: Petit exemple de mapping entre la facture Engie et la facture Energisme

- **business_values :** ce module contient tous les énumérations qui vont nous servir dans les autres module.
- **translator :** ce module contient la classe translator qui permet de faire la traduction. Il reçoit comme entrée une liste de factures d'un client (une ligne du fichier "getFactureDetails.json"), il va réaliser la traduction pour chaque facture de cette liste et renvoi parmi les factures traduites celle avec le plus grand nombre d'informations récupérer (le max de cette liste de factures traduite en fonction de leur longueur).
  Comme pour le précédent traducteur, il y a des méthodes qui renvoie une seule information, et il y a des méthodes qui retournent un ensemble d'informations (un dictionnaire clé valeur).

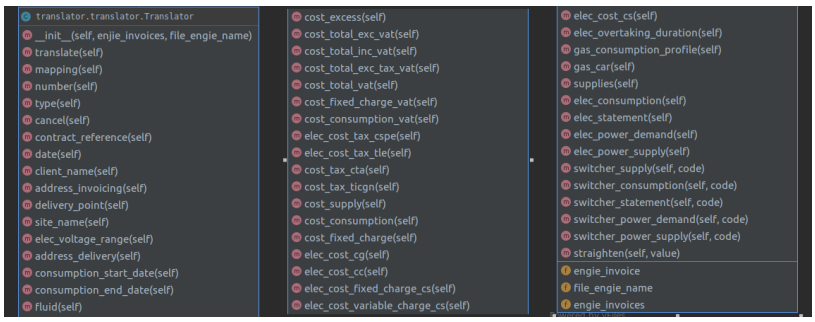


Figure 14: diagramme de classe de translator

- **progress :** ce module contient une classe qui me permet d'avoir une barre de chargement quand le programme est lancé, ça me permet de savoir le temps qui reste avant la fin d'exécution.

**package duplicate_detector:**

Après avoir fini le projet b2b_engie, il est utilisé afin de traduire les factures Engie et de produire des factures Energisme normalisés, sachant qu'il y avait déjà d'ancien factures traduites avec un autre mapping que le mien, donc cela à générer des doublons de facture dans leur base de données, il m'a donc fallu détecter et repérer ces doublons et de déterminer les différences entre ces factures afin de prendre une décision de celles qui vont être gardées et celles qui vont

être supprimées.
Ce package fait le boulot détaillé ci-dessus, il est constitué des modules suivants :

- **__main__ :** ce module contient le programme principal qui gère les arguments d'entrée et la fonction d'usage.
- **detector :** ce module contient une classe qui a un dictionnaire qui va garder tous les factures avec une clé unique (constitué du numéro de facture standardisé sans 0 au début + id du point de livraison).
  Pour les doublons, ils ont la même clé unique donc ils vont être ajouté dans la liste "invoices".
  À la fin, il suffit de faire un filtre par rapport à la longueur de la liste "invoices" et on trouve tous les doublons.
  Ce dictionnaire contient comme valeur pour chaque clé une liste de factures "invoices" la facture max, la facture min, les clés qui n'existent pas dans une facture et le plus important les valeurs différentes entre les mêmes clés.

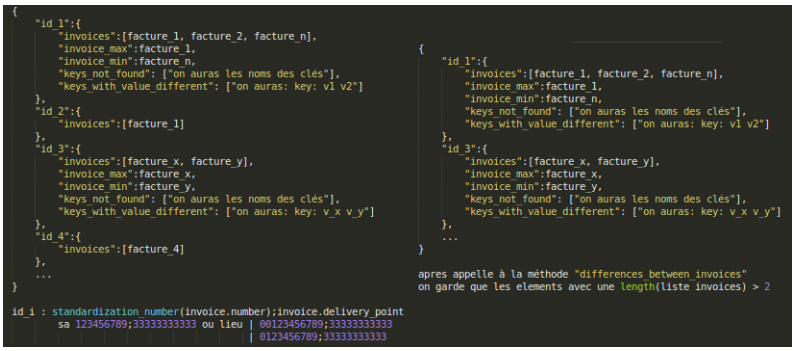


Figure 15: Représentation du detector

- **duplicate :** ce module contient les fonctions qui vont nous permettre d'avoir un système de "logging", de produire les déférents fichiers d'information et d'avoir comme résultat un répertoire par partenaire qui contient les fichiers d'informations :
  - list_duplicates.txt : contiens les invoice.number et le invoice.delivery_point afin de supprimer les doublons après.
  - list_differences.txt : contiens les différences entre les doublons.
  - list_length.txt : contiens les longueurs des différents doublons.

### 2.4.3 Conclusion

Je n'ai pas toujours appréhender le langage Python car je n'avais jamais codé avec, mais il s'avère qu'avec l'implémentation de ce projet, j'en ai conclu que c'est un langage de haut niveau facile à utiliser, capable de réaliser des choses complexes et sophistiqué, et cela en se basant sur peu de ligne de code. Pour moi, ce projet m'a permis d'acquérir la maîtrise d'un nouveau langage, et d'approfondir toutes les connaissances que j'ai pu acquérir auparavant. Cette première expérience m'a permis de :

- De réaliser une application python de "A" à "Z".
- De code avec les conventions python d'aujourd'hui.
- D'optimiser mon code en utilisant les lambdas expression.
- De réaliser une documentation et des tests unitaires pour toutes l'application.

## 2.5 b2b-meter (Java)

### 2.5.1 Introduction

Ce projet est un projet de l'entreprise déjà implémenté et utiliser, c'est un projet spring-boot, un client web service qui a comme ressource une requête post avec comme body un objet qui contient pleine d'information :

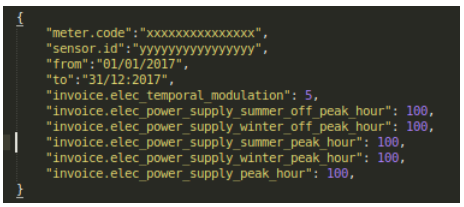


Figure 16: Exemple de l'objet injecté dans la requête post

Dans cette première version, l'information est injectée dans le body de la requête (car ils n'ont pas le b2b-Enids), pour qu'à la fin, on aurait comme résultat les puissances souscrites et les puissances demandées d'une date de début à une date de fin données. Tout cela sert dans le dessin des courbes de consommations et de prédictions. Le type de requête pour cette première version :
"www.site.com/"load_curve/_search
Le résultat, c'est un JSON qui contient :


Figure 17: Exemple de résultat de la requête

Ma contribution dans ce projet était de réaliser une deuxième version plus facile à utiliser et qui récupère les puissances souscrites réelles dans les objets "meter" dont le traducteur que j'ai implémenté "b2b-Enedis" génère a priori.
Le type de requête pour cette deuxième version :
"www.site.com/"load_curve/v2/{meter.code}/{from}/{to}
Le résultat, c'est un CSV qui contient :

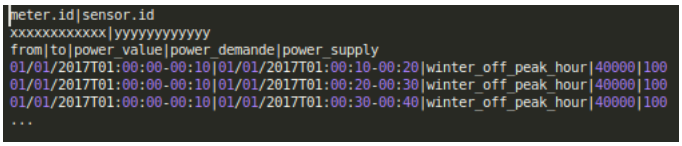


Figure 18: Exemple de résultat de la requête

### 2.5.2 Implémentation

Afin d'implémenter le besoin, j'ai procédé de la manier suivante, en gardant la même architecture que l'ancienne version :

- Implémentation de deux client web service avec feign :
  - Le premier celui qui reconnaît la requête load_curve/v2/{meter.code}/{from}/{to}

```
@GetMapping(value = "/v2/{meter.id}/{from}/{to}")
public ResponseEntity<String> searchV2(@PathVariable String meter_id,
                                       @PathVariable String from,
                                       @PathVariableString to) {
    LOGGER.debug("Get /{}/{}/{}", meter_id, from, to);
    LoadCurveReply result = loadCurveServiceV2.search(meter_id, from, to);
    return ok(result.toCSV());
}
```

Figure 19: Exemple de client de la deuxième version de load curve.

  - Le deuxième pour récupérer le bon "meter" dans la base de données.

```
@FeignClient(name = "synoptic", url = "${synoptic.url}")
public interface SynopticClient {
    @GetMapping( value = "/meter/{uid}")
    Map<String, Object> getMeterUid(@PathVariable("uid") String uid);
}
```

Figure 20: Interface de ce client.

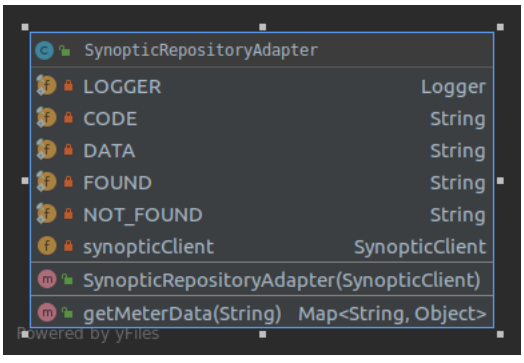


Figure 21: diagramme de classes de l'implémentation de ce client.

- Implémentation de toutes les autres classes avec les modifications nécessaires.

### 2.5.3 Conclusion

Ce projet est un projet complet et professionnel déjà utilisé, donc ça m'a permis d'avoir à quoi rassemble le code d'un vrai développeur qui a plus d'expérience que moi (10 ans). L'implémentation de ce web service m'a permis de :

- Créer une application web avec Spring.
- Créer des clients web service avec Feign.
- modifier un projet déjà opérationnel, afin de faire une deuxième version.

# 3 Conclusion générale

Ce stage a été très enrichissant pour moi, car il m'a permis de découvrir le monde professionnel, ainsi que les tâches quotidiennes d'un informaticien (développeur informatique). Il m'a permis de participer concrètement à ses enjeux au travers des missions de développement que j'ai réalisé. Durant mon stage j'ai pu mettre en pratique mes connaissances théoriques acquises durant ma formation de licence, tout en étant confronté aux difficultés réelles du monde du travail, en effet, Il y a sans cesse de nouvelles connaissance à apprendre et à acquérir et mon stage a été l'opportunité pour moi d'avoir une vue sur les déférents technologies (api, framework, architecture, etc.) utilisées dans le monde du développement informatique, cela m'a permis de me surpasser dans les missions qui m'ont été confiées.

Pour conclure après cette expérience, j'ai décidé de m'orienter et de m'engager beaucoup plus dans le domaine du développement informatique.