Centre IRSTEA de Clermont-Ferrand
9 avenue Blaise Pascal CS 20085

Rapport de projet d'élèves ingénieur
Filière 3 : Systèmes d'information et aide à la décision

Projet de robot d'indexation web de bulletins agricoles

Présenté par : Axel Lefèvre & Arnaud Olivier

Responsable ISIMA : Myoung-Ah Kang

Responsable de l'IRSTEA : Catherine Roussey
Stephan Bernard

Date de la soutenance : 16/02/017

Durée du projet : 12 semaines

Institut Supérieur d'Informatique, de Modélisation et de leurs Applications
www.isima.fr
Campus universitaire des Cézeaux • 1 rue de la Chebarde
TSA 60125 • CS 60026 • 63178 Aubière CEDEX
Tel (+33/0) 473 405 000 • Fax (+33/0) 473 405 001

Remerciements

Nous remercions l'Institut Supérieur d'Informatique, de Modélisation et de leurs Applications (ISIMA) ainsi que l'Institut national de Recherche en Science et Technologies pour l'Environnement et l'Agriculture (IRSTEA) pour nous avoir permis de réaliser ce projet.

Madame Catherine Roussey, pour son aide précieuse tout au long du projet et les conseils qu'elle nous a donné notamment pour la compréhension du contexte du projet.

Monsieur Stephan Bernard, pour sa disponibilité et son expérience technique qu'il a partagé avec nous afin d'apporter de la qualité à ce projet.

Table des figures

Figure 1 : Schéma des acteurs engagés dans la rédaction et la diffusion des BSV. 2
Figure 2 : Première page d'un BSV de la région Midi-Pyrénées concernant les noisettes 3
Figure 3 : Arborescence des BSV sur le site de la DRAAF d'Occitanie 4
Figure 4 : Exemple de script JavaScript utilisant la bibliothèque PhantomJS 7
Figure 5 : Exemple de paramétrage de Web Scraper 8
Figure 6 : Exemple d'arborescence avec Web Scraper 8
Figure 7 : Schéma représentant l'architecture de Scrapy 10
Figure 8 : Fichier CSV retourné par notre robot d'indexation 16
Figure 9 : Code HTML de la page http://draaf.occitanie.agriculture.gouv.fr/Bulletins-de-sante-du-vegetal 17
Figure 10 : Code HTML du site http://www.aquitainagri.fr/menu-horizontal/publications/bulletins-de-sante-du-vegetal-bsv.html 18
Figure 11 : Exemple des doublons dans le fichier BSV 20

Résumé

Le but de ce projet est de réaliser un **robot d'indexation** permettant de récupérer des **Bulletins de Santé du Végétal** (BSV) et les informations qui y sont associées, sur différents sites web. La réalisation de ce robot d'indexation nous a été confiée par Catherine Roussey et Stephan Bernard. Il s'agit d'un outil qui sera utilisé régulièrement afin de remplir une base de données des BSV et de la garder à jour.

Le développement de cet outil a été réalisé avec le langage de programmation **Python** avec notamment l'utilisation de la bibliothèque **Scrapy** qui contient de nombreuses fonctionnalités permettant la récupération d'informations sur un site web et leur traitement.

A ce jour, l'outil que nous avons développé est fonctionnel puisqu'il permet de récupérer les BSV sur différents sites mais nous n'avons pas pu traiter la totalité des sites publiant des BSV.

Mots-clés : Robot d'indexation, Bulletin de santé du végétal, Python, Scrapy.

Abstract

The goal of the project was to develop a **web crawler** allowing to collect "Bulletins de Santé du Végétal" (BSV) and associated information on different web site. The realization of this web crawler was entrusted to us by Catherine Roussey and Stephan Bernard. It is a tool which will be use regularly to fill a database and keep it up-to-date.

The development of this tool was carried with the programing language **Python** and his library **Scrapy** which contain many features to collect information on a web site and handle them.

Currently, the tool we have developed is functional since it allows the collect of BSV on different web site but we didn't manage to handle all of the site publishing the BSV.

Keywords : Web crawler, Python, Scrapy

Table des matières

Remerciements i
Table des figures ii
Résumé iii
Abstract iii
Table des matières iv
Introduction 1
I – Contexte du projet 2
A) Le Bulletin de Santé du Végétal (BSV) : 2
B) Distribution des BSV 4
C) Le besoin 5
II – Méthode de travail 6
A) Déroulement du projet 6
B) Les solutions possibles : 7
C) Scrapy 10
III – Les résultats 12
A) Notre solution 12
B) Les problèmes rencontrés 17
C) Les résultats obtenus 19
Conclusion 21
Référence bibliographique v
Glossaire vi
Annexes vii

Tous les mots qui sont suivis de * sont définis dans le glossaire.

Introduction

Ce projet nous a été confié par l'Institut national de Recherche en Science et Technologies pour l'Environnement et l'Agriculture (IRSTEA) et plus particulièrement par Catherine Roussey et Stephan Bernard. Nous étions également encadrés par Myoung-Ah Kang du côté de l'Institut Supérieur d'Informatique, de Modélisation et de leurs Applications (ISIMA).

Catherine Roussey et Stephan Bernard ont pour objectif de remplir une base de données indexée des Bulletins de Santé du Végétal (BSV). C'est dans ce cadre qu'ils nous ont confié la responsabilité de leur créer un outil leur permettant de récupérer la totalité des BSV ainsi que les informations qui y sont associés, sur différents sites web.

Ce projet est donc utile au remplissage de la base de données et sera utilisé régulièrement afin de garder de la garde à jour. En effet, jusqu'à présent, c'est Stephan Bernard qui s'occupait de récolter les BSV. Pour cela, il utilisait des scripts* lui permettant de récupérer les BSV sur les différents sites mais il devait changer les paramètres pour chacun des sites, et ça lui prenait beaucoup de temps. C'est donc dans l'optique d'améliorer les performances en termes de récupération des BSV et pour optimiser leur temps de temps de travail qu'ils nous ont confié ce projet.

Pour présenter ce projet, nous commencerons par détailler le contexte du projet et ses objectifs afin de comprendre ce que sont les BSV et où est ce que l'on peut les trouver. Puis, nous aborderons la façon dont nous avons travaillé pour répondre à leur besoin. Enfin, nous conclurons sur le travail réalisé, les objectifs atteints et les différents points importants du projet.

I – Contexte du projet

A) Le Bulletin de Santé du Végétal (BSV) :

De nos jours, le développement durable est une problématique qui touche chacun d'entre nous, afin de préserver notre environnement. C'est le cas notamment dans l'agriculture où l'on veut pouvoir continuer de cultiver des produits de qualité tout en préservant le sol et son écosystème. Pour cela, il faut réduire au maximum l'utilisation de pesticides et n'utiliser que ceux dont on a réellement besoin. On parle ainsi de durabilité agricole.

C'est dans cette optique que plusieurs entités gouvernementales et européennes ont mis en place des directives afin de préserver notre agriculture. En France, cela a commencé en 1943 avec les avertissements agricoles qui étaient rédigés par les Services Régionaux de la Protection des Végétaux (SRPV) pour les agriculteurs. Néanmoins, ces publications concernaient uniquement les cultures dites pérennes. A partir de 1960, les avertissements agricoles se sont étendus à d'autres types de cultures tels que les légumes. Mais en même temps, ils sont devenus payants sur abonnement.

Finalement, l'apparition des BSV tels qu'on les connaît aujourd'hui, est assez récente puisque les premières publications ont eu lieu en 2009 et redeviennent gratuites. Les BSV ont été mis en place par le Grenelle de l'environnement et le plan Ecophyto dans le but de renforcer les réseaux nationaux de surveillance sur les cultures et les pratiques agricoles. Ces publications sont rédigées par de nombreux rédacteurs (Figure 1) qui varient en fonction des régions. Ils s'appuient sur diverses informations telles que des analyses biologiques, des données météorologiques, des modèles épidémiologiques et des observations directement sur les cultures. Chaque année, il y a 150 000 parcelles qui sont observées afin que 4 000 rédacteurs puissent rédiger 3 400 BSV.

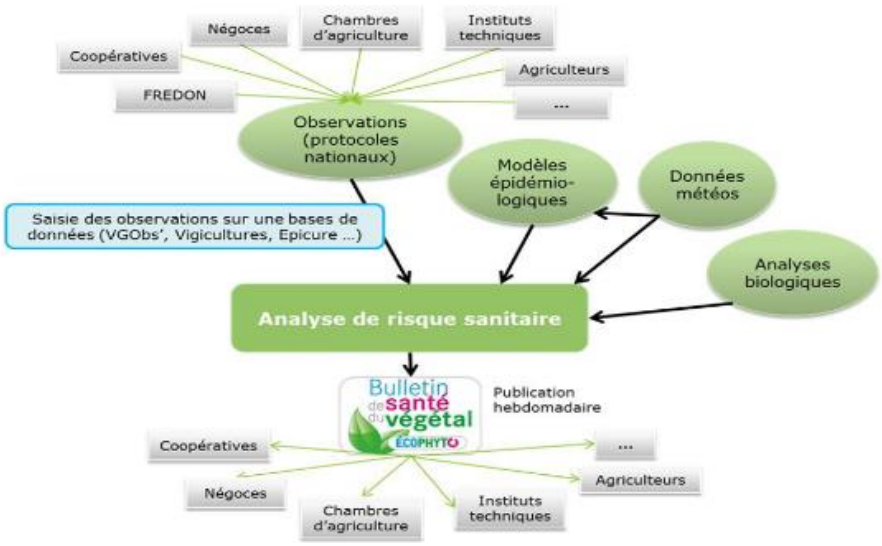

Figure 1 : Schéma des acteurs engagés dans la rédaction et la diffusion des BSV.

Du point de vue du contenu, le BSV (Figure 2) est une synthèse interprétée des observations décrites précédemment. Il a pour but de réunir les analyses techniques établies par les experts sur les risques phytosanitaires, mais aussi des informations sur la réglementation et son évolution, ainsi que des informations non réglementaires concernant des méthodes agricoles et la biologie afin que les personnes concernées par les BSV soient à jour dans leurs méthodes de travail.


Bulletin de Santé du Végétal

Aquitaine - Limousin - Poitou-Charentes - Midi-Pyrénées - Languedoc-Roussillon

Noisette

N°09

03/06/2016

Bulletin disponible sur www.aquitainagri.fr et sur le site de la DRAAF www.draaf.aquitaine-limousin-poitou-charentes.agriculture.gouv.fr

Recevez le Bulletin de votre choix GRATUITEMENT en cliquant sur : Formulaire d'abonnement au BSV

Ce qu'il faut retenir

- **Balanin :** L'accouplement des balanins est en cours. La période de ponte s'annonce bientôt.
- **Pucerons :** Les pucerons jaunes sont présents en verger. Le risque est élevé sans présence de la faune auxiliaire.
- **Acariens du feuillage :** La hausse des températures va entraîner une augmentation du risque dans les jours à venir.
- **Anthracnose à sphaceloma :** L'anthracnose est présente sur les pousses. Le risque est important
- **Adventices :** La présence d'adventices en vergers de noisetiers peut conduire à une concurrence spatiale importante de l'arbre et nuire à sa croissance ainsi qu'à sa productivité. Le risque est donc important surtout en jeunes vergers.

Bulletin de Santé du Végétal Aquitaine-Limousin-Poitou-Charentes–Midi-Pyrénées–Languedoc-Roussillon
Edition Aquitaine et Midi-Pyrénées
Noisette – N°09 du 03 Juin 2016

1/8

Figure 2 : Première page d'un BSV de la région Midi-Pyrénées concernant les noisettes

Ainsi, le BSV n'est pas un document qui préconise l'utilisation d'une méthode ou de certains pesticides. Ce document a pour objectif d'informer et d'améliorer les compétences et les savoirs des personnes concernées afin qu'ils puissent les appliquer en fonction de leurs besoins. Ainsi, les agriculteurs et autres collectivités peuvent utiliser les BSV pour cibler leur besoin et donc pour réduire leur coût en produits phytosanitaires tout en préservant leurs sols de toute pollution inutile.

B) Distribution des BSV

Depuis la parution des premiers BSV en 2009, ceux-ci sont accessibles gratuitement au format PDF en ligne. On les retrouve notamment sur les sites Internet des Chambres Régionales d'Agriculture (CRA), des Directions Régionales de l'Alimentation de l'Agriculture et de la Forêt (DRAAF) et aussi sur d'autres sites tiers. Il s'agit ici d'une première problématique puisqu'il n'existe pas de site référençant tous les BSV mais bien un site par région (on parle ici des nouvelles régions formées par la réforme territoriale de 2014). Ceci s'explique par le fait que les sols et les agresseurs peuvent varier d'une région à l'autre en fonction du climat, de même que les types de cultures par exemple.

Ensuite, un deuxième partitionnement se fait (Figure 3). En effet, chaque type de culture étant différent, il est nécessaire que chaque BSV soit spécifique à une culture afin d'être le plus précis possible. Selon les sites, il existe aussi d'autres répartitions supplémentaires, telles que des sous-zones géographiques correspondant aux anciennes régions ou aussi par année. Par exemple, sur le site de la DRAAF d'Occitanie, les BSV sont d'abord répartis par type de cultures puis par sous-région et enfin par année.

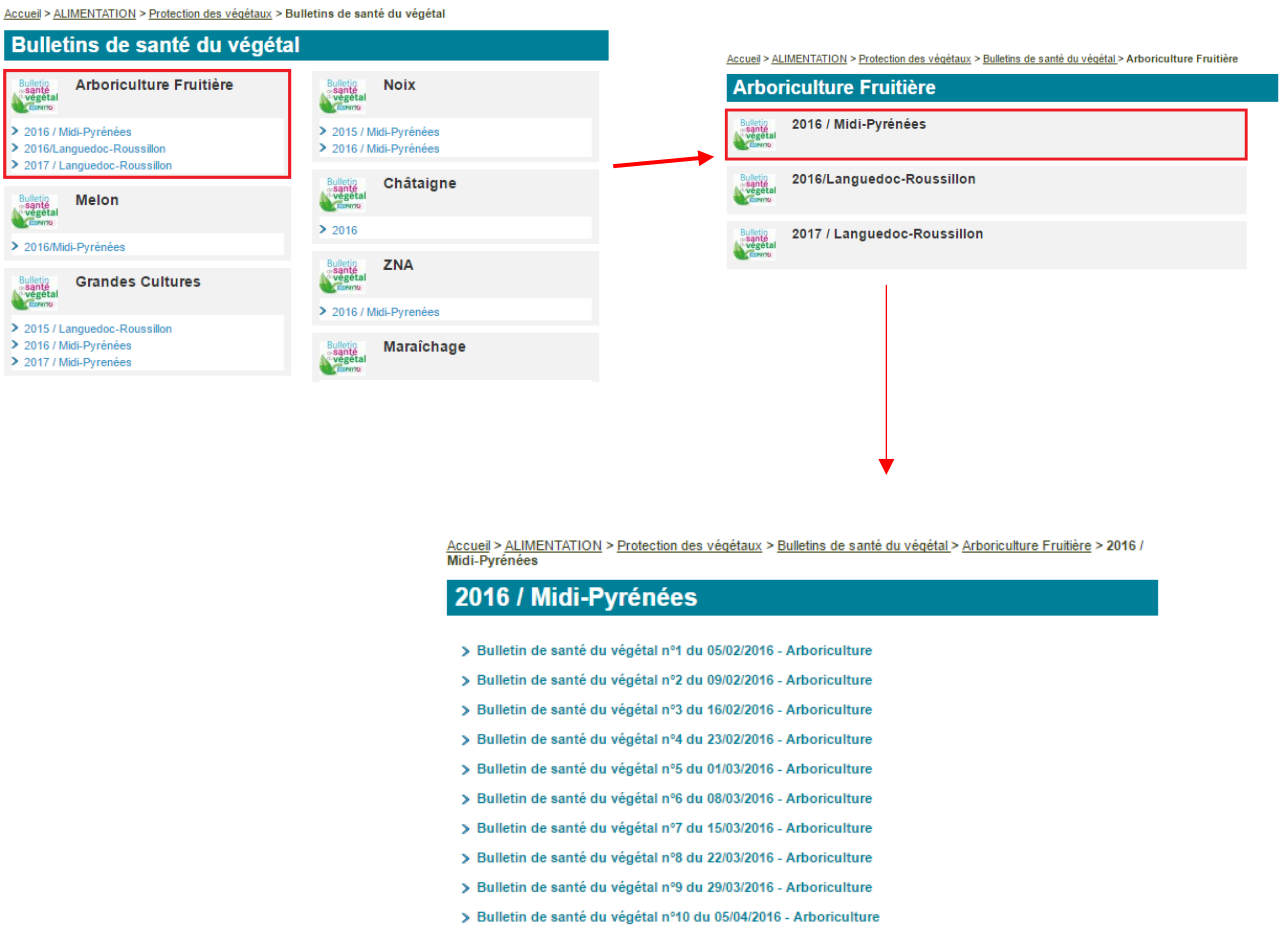

Figure 3 : Arborescence des BSV sur le site de la DRAAF d'Occitanie

C) Le besoin

Le but principal de ce projet est de pouvoir récupérer l'ensemble des BSV existants afin de remplir une base de données indexée selon certains critères tels que le type de cultures. Jusqu'à présent, la récupération des BSV était faite par des scripts qu'il fallait adapter à chaque site où se trouvaient les BSV. Le problème se pose alors sur la récupération de ces BSV. En effet, même s'il est possible de récupérer tous les BSV à un certain moment, il n'y a aucune règle quant à leur mise en ligne, ni aucun système d'alerte. Il devient donc difficile de garder la base de données à jour.

Le besoin de créer un outil à la fois automatique et efficace a donc émergé de ce problème. Afin de répondre à ce besoin, nous avons dû nous renseigner sur les outils libres de robot d'indexation existant. Un robot d'indexation est un outil permettant de collecter des informations sur une page web, comme par exemple une URL* ou un texte particulier. Le robot d'indexation aurait pour but de collecter les BSV sur les différents sites, avec en plus les informations associées (zone géographique, l'édition, le numéro, …). Pour cela, deux solutions se sont présentées à nous :

- Trouver un outil existant que l'on puisse paramétrer pour répondre à nos besoins ;
- Développer notre propre outil afin qu'il corresponde parfaitement à nos besoins.

De plus, la solution mis en place devait répondre ou au moins tendre vers certains critères.

Premièrement, la solution devra être sous licence libre. En effet, nous avions aucune restriction sur les technologies utilisées à partir du moment où leur licence était libre.

Ensuite, elle devra pouvoir être exécutée périodiquement sans intervention de l'utilisateur. Ici, on répond à un des problèmes soulevés. Ne sachant pas quand est-ce que de nouveaux BSV seront publiés, un lancement régulier et automatique de l'outil permet d'être le plus à jour possible sur la récupération des BSV, sans avoir à le faire vérifier par une personne.

Le point précédent peut entraîner un nouveau problème sur lequel il faut faire attention. En effet, si l'on veut que l'outil récupère périodiquement les nouveaux BSV publiés, on ne veut en aucun cas qu'il télécharge les BSV qui ont déjà été récupérés précédemment. On veut ainsi pouvoir éviter les doublons de fichiers inutiles afin de limiter l'espace de stockage des BSV et aussi éviter de surcharger le réseau sans aucune raison.

Pour contrer la multiplicité des sites où sont publiés les BSV, la solution devra être la plus générale possible, c'est à dire qu'il faut tendre vers une solution pour l'ensemble des sites plutôt que d'en avoir une par site. De cette manière, on peut potentiellement prévenir la modification d'un site ou l'ajout d'un nouveau puisque l'on aura au pire qu'à rajouter l'URL du nouveau site à indexer. Mais dans le cas où cela ne suffise pas, l'outil devra également pouvoir prévenir l'utilisateur qu'il ne peut pas récupérer les informations demandées afin que l'outil puisse être corrigé en fonction.

II – Méthode de travail

A) Déroulement du projet

Lors de la première réunion avec nos tuteurs, nous avons discuté en détail du projet et des objectifs ambitieux nous ont été fixés tout en sachant qu'ils pourraient difficilement être tenus. Ceci avait pour but d'anticiper certains besoins pour qu'on les prenne en compte même s'ils n'étaient pas réalisés. On assure ainsi la pérennité du projet tout en essayant de faciliter le travail de personnes qui finiront le projet. De plus, nous avons prévu de nous réunir régulièrement (environ toutes les deux semaines) afin de discuter de l'avancement du projet, des problèmes rencontrés et aussi pour assurer que nous ne dérivions pas du besoin initial.

Concernant le projet en lui-même (diagramme de GANNT en annexe), le travail s'est découpé en deux périodes. La première a été de comprendre le travail à réaliser et de trouver différentes solutions possibles pouvant répondre au besoin. En effet, nous n'avons pas été obligés d'utiliser une technologie en particulier mais plutôt de faire des recherches sur des solutions existantes sur les robots d'indexations. Pour cela, nous avons passé du temps à les essayer, à voir les avantages et inconvénients de chacune et bien sûr de vérifier qu'elles répondent aux mieux au besoin.

D'une fois ce travail de recherche réalisé, nous avons concentré nos efforts sur le développement du robot d'indexation. Pour cela, nous avons commencé par en développer un pour un seul des sites où se trouvent des BSV puis nous avons essayé de l'élargir au maximum le nombre de sites que notre robot pouvait indexer dans le but de répondre au besoin de généralisation de la solution. En parallèle à cela, nous avons rédigé le rapport et préparer la soutenance.

Par rapport au planning initialement prévu, la plus grosse différente s'est faite au niveau du développement. En effet, nous avons rencontré plusieurs problèmes qui seront détaillés ultérieurement. La conséquence a été que nous n'avons pas pu aller aussi loin que nous le voulions dans le développement de la solution notamment sur le nombre de sites que notre robot d'indexation peut gérer.

B) Les solutions possibles :

Comme expliqué dans la partie précédente, le projet a commencé par un travail de recherches sur les outils de robot d'indexation existant. Pour cela nous pouvions, soit trouver une solution toute faite que l'on peut adapter à notre besoin, soit créer notre propre solution à l'aide d'outil prévu pour. Typiquement des bibliothèques de développement associées à un langage de programmation. Lors de nos recherches, nous avons trouvé 3 solutions qui potentiellement pouvaient répondre à nos besoins :

- PhantomJS : Développé par Ariya Hidaya, il s'agit d'un navigateur web headless (sans interface graphique) scriptable utilisé pour automatiser des interactions avec des pages web. Il s'agit d'une API (Application Programming Interface) JavaScript* qui permet de pouvoir faire des scripts (Figure 4) pour automatiser la navigation sur des sites web afin d'y récupérer des informations. Il permet aussi de faire des captures d'écran des sites web, de simuler des comportements utilisateurs, de réaliser des tests... C'est un outil très complet, que l'on peut adapter à nos besoins puisqu'en plus de PhantomJS, on a la possibilité d'utiliser les bibliothèques standard de JavaScript. Néanmoins, l'utilisation de PhantomJS nécessite de savoir utiliser JavaScript correctement, ce qui n'est pas notre cas. C'est donc pour cette raison que nous avons éliminé PhantomJS des solutions possibles pour répondre à notre besoin.

```
1  var page = new WebPage();
2
3  page.onConsoleMessage = function(msg) {
4      console.log(msg);
5  };
6
7  page.open(encodeURI("http://mobile.twitter.com/JSZurich"), function (status) {
8      if (status !== "success") {
9          console.log("Unable to access network");
10     } else {
11         page.evaluate(function() {
12             var list = document.querySelectorAll('span.status');
13             for (var i = 0; i < list.length; ++i) {
14                 console.log((i + 1) + ": " + list[i].innerHTML.replace(/<.*?>/g, ''));
15             }
16         });
17     }
18     phantom.exit();
19 });
```

Figure 4 : Exemple de script JavaScript utilisant la bibliothèque PhantomJS

- Web Scrapper : C'est une entreprise spécialisée dans l'extraction de données sur des sites web qui proposent un service payant pour les entreprises et aussi une extension Google Chrome gratuite. Cette extension permet de récupérer des informations sur un site Web et d'en faire un fichier CSV*. La prise en main de l'outil est assez rapide puisqu'on utilise directement dans Google Chrome, dans les « outils pour développeurs ». A partir de cette interface, il suffit d'entrer l'URL du site web où l'on veut récupérer les informations puis, on peut directement sélectionner (Figure 5) les éléments de la page web pour que l'outil les récupère.

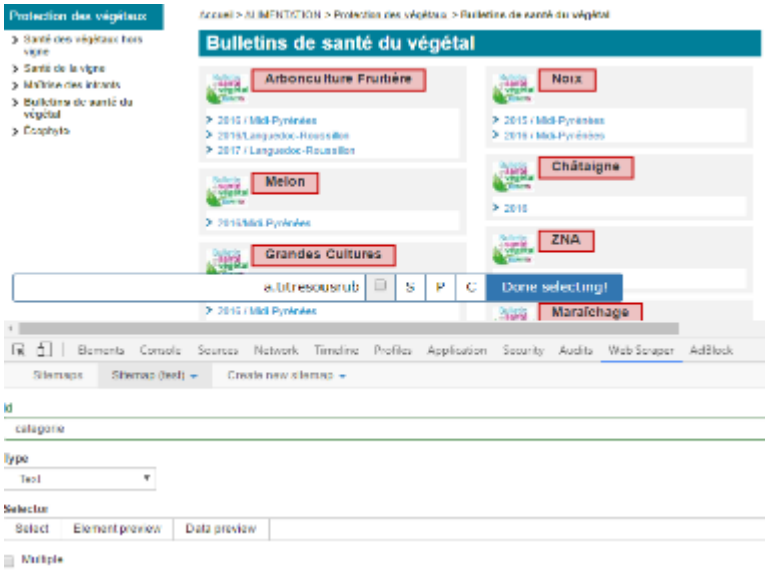

Figure 5 : Exemple de paramétrage de Web Scraper

De cette façon, on peut créer une arborescence (Figure 6) dans les informations que l'on veut récupérer pour que l'outil puisse mettre en place les informations correctement dans le fichier CSV. De plus, on peut sauvegarder le paramétrage fait pour chaque site et l'exporter sur une autre machine.

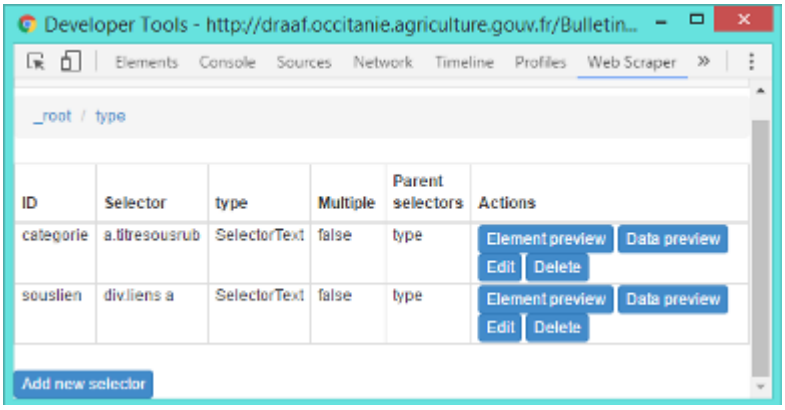

Figure 6 : Exemple d'arborescence avec Web Scraper

Les avantages de cette solution sont la simplicité d'utilisation, la facilité de maintien et la réduction considérable de la phase de développement. Néanmoins, cela implique tout de même de faire un paramétrage de l'outil pour chacune des pages concernées car il ne permet pas de naviguer d'une page à une autre. De plus, nous sommes bridés par les fonctionnalités de l'outil, notamment si l'on veut qu'il s'exécute automatiquement ou pour nous prévenir en cas d'erreur.

- Scrapy : Il s'agit d'un framework* Python open-source permettant la création de robots d'indexation qui, eux permettent l'extraction de données sur des sites web. Ce framework dispose d'une forte communauté offrant une grande robustesse, et il est aussi régulièrement enrichi de nouvelles fonctionnalités. De plus, le code à générer est assez court étant donné que de nombreuses opérations sont gérées par Scrapy directement. Ainsi, on peut facilement récupérer des données et les sauvegarder dans un fichier sous différents formats, tels que le format CSV. On peut aussi réutiliser les informations récoltées comme les URL pour passer d'une page web à une autre et relancer un nouvel algorithme sur cette nouvelle page. C'est en partie pour ces raisons que nous avons choisi d'utiliser Scrapy pour réaliser notre robot d'indexation dont le fonctionnement plus approfondie est expliquée dans la partie suivante.

C) Scrapy

Le schéma suivant (figure 7) montre une vue d'ensemble de l'architecture Scrapy avec ses composants et un aperçu des flux de données qui ont lieu à l'intérieur du système (indiqué par les flèches rouges). Les flux de données sont également décrits ci-dessous. Par mesure de clarté, nous utiliserons les mots anglais utilisés dans le schéma pour le décrire.

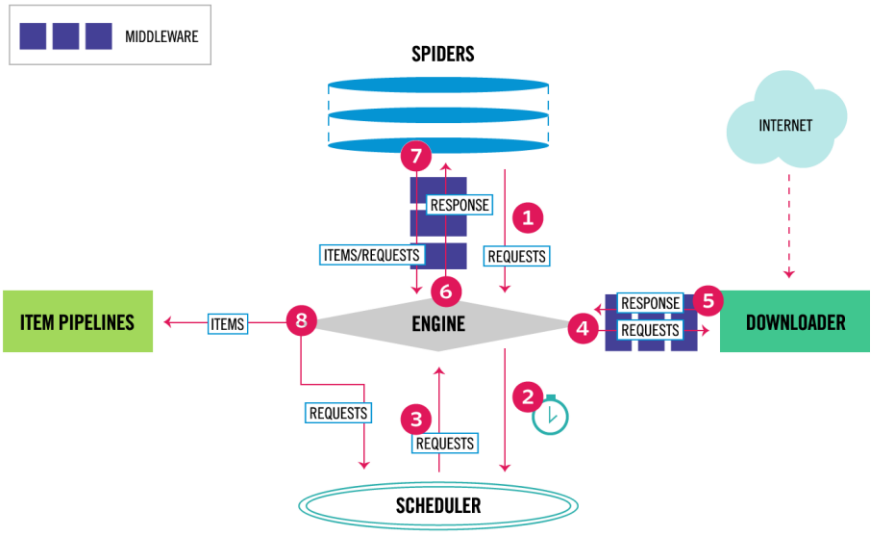

Figure 7 : Schéma représentant l'architecture de Scrapy

Voici le détail des différents composants et leur utilité au sein du système :

- Engine : il est chargé de contrôler le flux de données entre tous les composants du système et de déclencher des événements si certaines actions se produisent
- Scheduler : il reçoit des requêtes de la part de l'Engine et les stocks pour les renvoyer plus tard lorsque l'Engine en fera la demande.
- Downloader : il est responsable de la récupération des pages web et de renvoyer une réponse en conséquence à l'Engine.
- Spiders : ce sont des classes personnalisées écrites par les utilisateurs Scrapy pour analyser les réponses et les éléments d'extraction de leurs requêtes ou des requêtes supplémentaires à suivre.
- Item Pipeline : il est responsable du traitement des données une fois qu'elles ont été extraites par les Spiders. Les tâches typiques incluent le nettoyage, la validation et la persistance (comme le stockage d'un élément dans une base de données).

- Downloader middlewares : Les Downloader middlewares sont des liens spécifiques qui se situent entre l'Engine et le Downloader et qui traitent les requêtes quand elles passent de l'Engine au Downloader, et les réponses qui passent du Downloader à l'Engine.
- Spider middlewares : Spider middlewares sont des liens spécifiques qui se situent entre l'Engine et les Spiders et sont en mesure de traiter les entrées (réponses) et sorties (données et requêtes) d'un Spider.

Ainsi, les flux de données sont contrôlés par l'engine de la façon suivante :

1) L'Engine reçoit les requêtes initiales d'extraction de données de la part du Spider.
2) L'Engine planifie les requêtes dans le scheduler et demande quelle est la prochaine requête à traiter.
3) Le scheduler renvoie la prochaine requête à traiter à l'engine.
4) L'engine envoie la requête au downloader en passant par les downloader Middlewares.
5) Une fois que la page a fini d'être téléchargée, le downloader créée et renvoie une réponse à l'engine en passant par les downloader middlewares.
6) L'engine reçoit la réponse et l'envoie au spider pour la traiter, en passant par le spider middleware.
7) Le spider traite la réponse et retourne les données récupérées ainsi que les nouvelles requêtes, en passant par le spider middleware.
8) L'engine envoie les données récupérées à l'item pipelines, puis envoie les requêtes traitées au scheduler et demande s'il y a des nouvelles requêtes à traiter.
9) Le processus se répète (depuis l'étape 1), jusqu'à ce que le scheduler ne renvoie plus de nouvelle requête.

III – Les résultats

A) Notre solution

Le framework que nous utilisons est compatible avec Python 2.7. Sur machine Linux il est conseillé d'utiliser une machine virtuelle Python telle que virtualenv pour empêcher les conflits avec les packages Pyhton déjà installés. Pour lancer une session sur virtualenv, il suffit d'entrer la commande suivante : **source bin/activate** dans le terminal en se positionnant à la racine du dossier qui possède l'environnement.

Un projet Scrapy se présente sous la forme de plusieurs fichiers. Voici ceux de notre programme que l'on a testé sur le site :
http://draaf.occitanie.agriculture.gouv.fr/Bulletins-de-sante-du-vegetal

- Un fichier **settings.py** qui permet de régler différents paramètres de notre logiciel comme par exemple vers quel fichier et de quel type vont être stockées les données ou ressources.

```
# Scrapy settings for mycrawler project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/topics/settings.html
#

BOT_NAME = 'mycrawler'

SPIDER_MODULES = ['mycrawler.spiders']
NEWSPIDER_MODULE = 'mycrawler.spiders'

DOWNLOAD_DELAY = 5


FEED_URI = 'item.csv'

FEED_FORMAT = 'csv'
FEED_EXPORTERS_BASE = {
'csv': 'scrapy.contrib.exporter.CsvItemExporter',
}

ROBOTSTXT_OBEY = True
```

Ici le fichier de stockage de données sera au format csv et s'appellera **item.csv**.

- Un fichier **pipelines.py** qui peut permettre d'ajouter une base de données comme stockage des ressources ou données collectées.

```
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/topics/item-pipeline.html

class MycrawlerPipeline(object):
    def process_item(self, item, spider):
        return item


#class MySQLPipeline(object):
#    def __init__(self):
#        import MySQLdb
#        self.db = MySQLdb.connect(host="localhost", user="root",
passwd="", db="crawler_engine", charset = 'utf8', use_unicode = False)
#
#    def process_item(self, item, spider):
#        cursor = self.db.cursor()
#        for i in range(len(item['link'])):
#            cursor = self.db.cursor()
#            sql = "insert into urls(url, domain, num_crawl) values
('%s','%s','%s')" % (item['link'][i],'probikeshop', 1)
#            cursor.execute(sql)
#            self.db.commit()
#        return item
```

Ici le **pipelines.py** du projet avec en commentaire un exemple d'utilisation avec MySQL. Il permet d'enregistrer les informations que le robot d'indexation à récupérer, dans une base de données.

- Un fichier **items.py** où l'on va définir les données ou ressources que l'on veut collecter.

```
# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/topics/items.html

from scrapy.item import Item, Field

class MycrawlerItem(Item):
      title = Field()
      url = Field()
      pdf = Field()
```

Ici on déclare que l'on va collecter des données qui seront classées comme **title**, **url** ou **pdf**.

- Un fichier **nom_de_mon_crawler.py** qui contient le code qui va permettre de définir à partir d'où et de quelle manière collecter les données et ressources.

```
import scrapy
from scrapy.spider import BaseSpider
from scrapy import Selector
from mycrawler.items import MycrawlerItem
from urlparse import urljoin



#Définition du domaine et des URLs à parcourir
class MycrawlerSpider(BaseSpider):
    name = "mycrawler"
    allowed_domains = ["draaf.occitanie.agriculture.gouv.fr"]
    base = "http://draaf.occitanie.agriculture.gouv.fr"
#Base que l'on utilisera pour rentrer dans des liens
    start_urls = ["http://draaf.occitanie.agriculture.gouv.fr/Bulletins-de-
sante-du-vegetal"]




#Déclaration des différents moyen de parcours et d'extraction des liens

    #Parse qui va récupérer tout les liens vers les types de culture
    def parse(self, response):
            hxs = Selector(response)
            titles = hxs.xpath('//div[@class="listerub"]')
#On va s'intéresser aux données comprises dans div[@class="listerub"]
            for title in titles:
                item = MycrawlerItem()
#Les items recherchés sont ceux de items.py
                item['url'] = urljoin(self.base,
''.join(title.xpath('a[@class="titresousrub"]/@href').extract()))
#URL vers la page d'un type de culture
                yield scrapy.Request(item['url'], self.parse_type)
#Retourne la réponse de parse_type sur le lien URL sélectionné
```

```
    #Parse qui va récupérer tout les liens vers les regions et années d'un
type de culture
    def parse_type(self, response):
            hxs = Selector(response)
            titres = hxs.xpath('//div[@class="rub"]')
#Sert pour récupérer le nom des types de culture
            titles = hxs.xpath('//div[@class="listerub"]')
#Sert pour récupérer les liens vers les régions et années
            items = []   #On initialise le tableau items à NULL
            for titre in titres:
                  item = MycrawlerItem()
                  item['title'] = titre.xpath('h2/text()').extract()
#On récupère le nom de type de culture
                  item['url'] = response.url
#On récupère l'URL de la page où l'on se trouve
                  items.append(item)
#On ajoute les données récupérées dans le tableau items
            for title in titles:
                  item = MycrawlerItem()
                  item['url'] = urljoin(self.base,
''.join(title.xpath('a[@class="titresousrub"]/@href').extract()))
#URL vers la page d'une région et année
                  request = scrapy.Request(item['url'],
self.parse_extraction)
#Requête qui execute parse_extraction sur le lien URL sélectionné
                  request.meta['items'] = items
#Garde les données du tableau items dans la requête
                  yield request



    #Parse qui récupère le nom de la région avec son année et les URLs des
BSV
    def parse_extraction(self, response):
        hxs = Selector(response)
            titles = hxs.xpath('//div[@class="listeart"]')
#Sert pour récupérer le nom de la région et son année
            titres = hxs.xpath('//div[@class="rub"]')
#Sert pour récupérer les BSV
            new_items = response.request.meta['items']
#Récupère les données du tableau items
            for titre in titres:
              new_item = MycrawlerItem()
              new_item['title'] = titre.xpath('h2/text()').extract()
#On récupère le nom de la région et son année
                  new_item['url'] = response.url
#On récupère l'URL de la page où l'on se trouve
                  new_items.append(new_item)
#On ajoute les données récupérées dans le tableau new_items
            for title in titles:
            new_item = MycrawlerItem()
                  new_item['title'] = title.xpath('a/text()').extract()
#On récupère l'intitulé de la BSV
                  new_item['url'] = urljoin(self.base,
''.join(title.xpath('a/@href').extract())) #On récupère l'URL de la BSV
                  new_item['pdf'] = "pdf"
#On précise que l'URL donne bien sur un PDF
                  new_items.append(new_item)
#On ajoute les données récupérées dans le tableau new_items
        return new_items #On retourne les données récupérées
```

D'abord on importe depuis les bibliothèques Scrapy les méthodes que l'on va utiliser. Puis on définit le domaine et les URL que l'on va explorer. Enfin, on définit différentes méthodes (ici « parse », « parse_type » et « parse_extraction »), qui correspondent aux différents moyens de parcours et d'extraction de données d'une page. La fonction « **parse »** est lue en première par défaut.

Le programme donne ainsi un fichier CSV de la forme suivante (figure 8) :

http://draaf.occitanie.agriculture.gouv.fr/Ail,351		Ail
http://draaf.occitanie.agriculture.gouv.fr/2017-Midi-Pyrenees,488		2017 / Midi-Pyrénées
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no1,1480	pdf	Bulletin de santé du végétal n°1 du 02/12/2017 - Ail
http://draaf.occitanie.agriculture.gouv.fr/2016,352		2016 / Midi-Pyrénées
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no1,443	pdf	Bulletin de santé du végétal n°1 du 17/12/2015 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no3,649	pdf	Bulletin de santé du végétal n°3 du 11/3/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no2,587	pdf	Bulletin de santé du végétal n°2 du 19/2/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no4,666	pdf	Bulletin de santé du végétal n°4 du 18/3/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no5,720	pdf	Bulletin de santé du végétal n°5 du 31/3/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no6,773	pdf	Bulletin de santé du végétal n°6 du 07/04/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no7,814	pdf	Bulletin de santé du végétal n°7 du 14/04/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no8,838	pdf	Bulletin de santé du végétal n°8 du 22/04/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no9,857	pdf	Bulletin de santé du végétal n°9 du 28/04/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no10,924	pdf	Bulletin de santé du végétal n°10 du 13/05/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no11,943	pdf	Bulletin de santé du végétal n°11 du 19 /05/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no12,980	pdf	Bulletin de santé du végétal n°12 du 2619 /05/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no13,1054	pdf	Bulletin de santé du végétal n°13 du 09/06/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-HORS	pdf	Bulletin de santé du végétal HORS-SERIE du 24/10/2016 - Ail
http://draaf.occitanie.agriculture.gouv.fr/Arboriculture-Fruitiere		Arboriculture Fruitière
http://draaf.occitanie.agriculture.gouv.fr/2017-Languedoc-Roussillon		2017 / Languedoc-Roussillon
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no1,1544	pdf	Bulletin de santé du végétal n°1 du 18/01/2017 - Arboriculture
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no2,1571	pdf	Bulletin de santé du végétal n°2 du 02/02/2017 - Arboriculture
http://draaf.occitanie.agriculture.gouv.fr/2016-Languedoc-Roussillon		2016/Languedoc-Roussillon
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no15,1259	pdf	Bulletin de santé du végétal n°15 du 04/08/2016 - Arboriculture
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no7,794	pdf	Bulletin de santé du végétal n°7 du 13/04/2016 - Arboriculture
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no6,799	pdf	Bulletin de santé du végétal n°6 du 30/05/2016 - Arboriculture
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no8,869	pdf	Bulletin de santé du végétal n°8 du 27/04/2016 - Arboriculture
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no9,973	pdf	Bulletin de santé du végétal n°9 du 25/05/2016 - Arboriculture
http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no10,1167	pdf	Bulletin de santé du végétal n°10 du 25/05/2016 - Arboriculture

Figure 8 : Fichier CSV retourné par notre robot d'indexation

La première colonne correspond aux URL récupérées, la deuxième indique si l'URL correspond à un PDF et la troisième colonne correspond aux informations au sujet des URL.

B) Les problèmes rencontrés

Durant la réalisation du projet, nous avons été confrontés à différents problèmes techniques. La principale difficulté a eu lieu lors de l'installation du Framework Scrapy. En effet, nous avons travaillé sur deux ordinateurs différents mais la solution que nous développions sur un ne fonctionnait pas sur l'autre et même si notre solution pouvait fonctionner, il a fallu déterminer les raisons pour lesquelles cela ne fonctionnait pas partout. Pour cela, nous avons commencé par tester différents paramètres d'exécution de notre solution mais cela ne changeait strictement rien. Il s'agissait en fait d'un problème à l'installation du framework. En effet, selon le système d'exploitation utilisé des conflits pouvaient avoir lieu entre les fichiers d'installation de Scrapy et les fichiers déjà installés de Python. Pour répondre à ce problème, il y a deux solutions. La première est d'installer Scrapy uniquement pour l'utilisateur qui va l'utiliser plutôt que de faire l'installation sur la totalité du système. Il se peut que cette première solution ne fonctionne pas, dans ces cas-là, il faut au préalable installer un environnement virtuel* pour python. Ceci permet d'avoir un environnement isolé pour l'installation de Scrapy et ainsi éviter le conflit entre les fichiers.

Le deuxième problème auquel nous avons été confrontés a eu lieu pendant le développement du robot d'indexation. En effet, nous utilisons le code HTML* de la page pour identifier les éléments que l'on veut récupérer, or si certains sites utilisent la même logique dans l'écriture de leur page HTML, ce n'est pas le cas de tous. Ceci représente un réel problème étant donné que nous voulons un seul robot d'indexation pour l'ensemble des sites. Pour illustrer ceci, prenons comme exemple la page des BSV de la DRAAF d'Occitanie et la même page de la CRA d'Aquitaine.

Figure 9 : Code HTML de la page http://draaf.occitanie.agriculture.gouv.fr/Bulletins-de-sante-du-vegetal

Comme le montre l'illustration précédente (Figure 9), la DRAAF d'Occitanie utilise des balises* « div », elles-mêmes composées de balises « img » et « a » afin de catégoriser les différents BSV. En revanche, le site de la CRA d'Aquitaine (Figure 10) utilise des balises « li » avec aussi des balises « a ». Ces différences entre chaque site complexifient la possibilité d'avoir un robot d'indexation efficace pouvant traiter un maximum de sites. C'est pour cela que nous avons décidé de traiter en priorité tous les sites de la DRAAF qui ont une architecture* similaire et qui contiennent une assez grande partie des BSV. Par manque de temps nous n'avons pas pu en traiter d'autres.

Figure 10 : Code HTML du site http://www.aquitainagri.fr/menu-horizontal/publications/bulletins-de-sante-du-vegetal-bsv.html

C) Les résultats obtenus

A la fin du projet, nous avons donc obtenu un robot d'indexation qui permet de récupérer les BSV de plusieurs sites de la DRAAF. La raison pour laquelle cela ne fonctionne que sur certains sites vient principalement de la diversité de l'architecture des sites.

Le programme pourrait être amélioré pour pouvoir parcourir d'autres pages web de la DRAAF qui ont à peu près la même structure. Pour cela, il faudrait lister tous les types de classe des différentes pages pour qu'elles puissent être toutes lues par le même programme.

Il faudrait faire varier aussi l'adresse de base du site (nom de domaine) selon l'URL d'origine qui est parcouru, car on s'en sert pour collecter les données d'une page à une autre, mais les différents sites de la DRAAF n'ont pas les mêmes noms de domaine d'URL. L'utilisation d'un tableau de noms de domaine semble envisageable.

Il existe aussi le problème de type de culture qui ne passe par une page en moins ou en plus par rapport à d'habitude pour donner leurs BSV. Il faudrait ainsi lister ces différents cas particuliers et leur appliquer une règle d'extraction de données à part.

De plus, nous avons essayé de répondre à un maximum des besoins énoncés au début du projet.

Tout d'abord, comme il nous a été demandé, la solution rendue est sous licence libre.

Ensuite, le robot d'indexation est exécutable en ligne de commande. On peut donc facilement paramétrer une exécution régulière à l'aide de scripts par exemple.

Nous avons aussi essayé d'anticiper une méthode de dédoublonnage en sauvegardant les résultats du robot d'indexation dans un fichier au format CSV. En effet, ce format de fichier permet facilement de traiter les données qu'il contient. Ainsi, on peut imaginer avoir un autre fichier CSV implémenté de la même façon et contenant les BSV déjà récupérés. Il suffirait alors simplement de comparer les fichiers et trouver ceux qui ont déjà été téléchargés.

Le dernier point à traiter est le suivant. Dans le fichier lui-même, on remarquera que plusieurs lignes sont identiques (figure 11), cela est dû à la méthode **parse_type** qui fait appel à **parse_extraction** en gardant les données dans **items**. En effet, **parse_type** va lister les URL des régions et années d'un type de culture, puis **parse_extraction** collecter le nom de la région avec année et les BSV, en mettant à jour le tableau **items** et en retournant les données. Ainsi lorsque **parse_type** appel **parse_extraction** une autre fois, les premières données vont rester dans **items** et vont être retournées dans le fichier CSV une autre fois, d'où les doublons.

Ainsi le procédé s'opère comme tel :

- items = [ ]
- items = [type de culture], **parse_type -> parse_extraction**
- items = [type de culture; région1; liste de BSV1], **insère** dans le fichier CSV
- items = [type de culture; région1; liste de BSV1], **parse_type -> parse_extraction**
- items = [type de culture; région1; liste de BSV1;région2;liste de BSV2], **insère** dans le fichier CSV

Ainsi les données de type de culture, région1 et liste de BSV1 apparaissent deux fois dans le fichier CSV.

152	http://draaf.occitanie.agriculture.gouv.fr/Arboriculture-Fruitiere		Arboriculture Fruitière
153	http://draaf.occitanie.agriculture.gouv.fr/2017-Languedoc-Roussillon		2017 / Languedoc-Roussillon
154	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no1,1544	pdf	Bulletin de santé du végétal n°1 du 18/01/2017 - Arboriculture
155	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no2,1571	pdf	Bulletin de santé du végétal n°2 du 02/02/2017 - Arboriculture
156	http://draaf.occitanie.agriculture.gouv.fr/Arboriculture-Fruitiere		Arboriculture Fruitière
157	http://draaf.occitanie.agriculture.gouv.fr/2017-Languedoc-Roussillon		2017 / Languedoc-Roussillon
158	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no1,1544	pdf	Bulletin de santé du végétal n°1 du 18/01/2017 - Arboriculture
159	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no2,1571	pdf	Bulletin de santé du végétal n°2 du 02/02/2017 - Arboriculture
160	http://draaf.occitanie.agriculture.gouv.fr/2016-Languedoc-Roussillon		2016/Languedoc-Roussillon
161	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no15,1259	pdf	Bulletin de santé du végétal n°15 du 04/08/2016 - Arboriculture
162	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no7,794	pdf	Bulletin de santé du végétal n°7 du 13/04/2016 - Arboriculture
163	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no6,799	pdf	Bulletin de santé du végétal n°6 du 30/05/2016 - Arboriculture
164	http://draaf.occitanie.agriculture.gouv.fr/Bulletin-de-sante-du-vegetal-no8,869	pdf	Bulletin de santé du végétal n°8 du 27/04/2016 - Arboriculture

Figure 11 : Exemple des doublons dans le fichier BSV

Conclusion

Notre objectif était de construire entièrement un outil permettant de récupérer les BSV ainsi que leurs informations associées sur différents sites web, tout en ayant pour but d'anticiper certains problèmes qui pouvaient se poser. Notamment, le fait qu'on ne veuille pas récupérer deux fois le même BSV.

Les objectifs ont été partiellement atteints puisque notre outil ne permet pas de récupérer la totalité des BSV. Ceci est dû à la diversité des sites qui distribuent les BSV et nous avons manqué de temps pour réussir à tous les traiter. Néanmoins, cela laisse plusieurs perspectives d'améliorations. La première bien évidement serait de pouvoir traiter la totalité des sites distribuant les BSV et il pourrait être aussi intéressant d'intégrer un système d'anti-doublon directement au moment où l'on récolte les BSV.

Ce projet a été pour nous l'occasion d'enrichir nos compétences techniques puisque nous avons dû faire des recherches sur différentes technologies telles que JavaScript et Python que nous n'avions pas ou très peu utilisé auparavant. Dans une moindre mesure, nous avons aussi pu nous initier sur les enjeux d'une agriculture responsable et de l'aménagement durable des territoires.

Référence bibliographique

[1] http://agriculture.gouv.fr/bulletins-de-sante-du-vegetal (02/02/2016)

[2] http://phantomjs.org/quick-start.html (Décembre 2016)

[3] http://webscraper.io/documentation (Décembre 2016)

[3] https://doc.scrapy.org/en/latest/intro/tutorial.html (Janvier 2017)

[4] https://doc.scrapy.org/en/latest/intro/install.html (Janvier 2017)

[5] https://freres.peyronnet.eu/tutoriel-scrapy/ (Janvier 2017)

[6] http://stackoverflow.com/questions/tagged/scrapy (Janvier 2017)

[7] http://lebouquetin.github.io/scrapy-presentation-pyuggre-01-2015/#/section-1/page-1 (Janvier 2015)

[8] https://doc.scrapy.org/en/latest/topics/architecture.html (Janvier 2017)

Glossaire

Architecture : En informatique, architecture désigne la structure générale inhérente à un système informatique, l'organisation des différents éléments du système (logiciels et/ou matériels et/ou humains et/ou informations) et des relations entre les éléments.

CSV : Comma-separated values, connu sous le sigle CSV, est un format informatique ouvert représentant des données tabulaires sous forme de valeurs séparées par des virgules.

Framework : En programmation informatique, un framework ou structure logicielle est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.

HTML : L'HyperText Markup Language, généralement abrégé HTML, est le format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et logiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des programmes informatiques.

JavaScript : JavaScript est un langage de programmation orienté objet de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs.

Script : Un langage de script est un langage de programmation qui permet de manipuler les fonctionnalités d'un système informatique configuré pour fournir à l'interpréteur de ce langage un environnement et une interface qui déterminent les possibilités de celui-ci.

URL : Le sigle URL (de l'anglais Uniform Resource Locator, littéralement « localisateur uniforme de ressource »), auquel se substitue informellement l'expression adresse web, désigne une chaîne de caractères utilisée pour adresser les ressources du World Wide Web.

Annexes

Annexe 1 : GANTT réel du projet I

Annexe 1 : GANTT réel du projet