École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
4e année
2011 - 2012

Rapport de projet Science de la Décision

Pré-traitement de données publiques Open Data

Encadrants
Gilles VENTURINI
gilles.venturini@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Rémi CARUYER
remi.caruyer@etu.univ-tours.fr
Pierre FOURREAU
pierre.fourreau@etu.univ-tours.fr

DI4 2011 - 2012

Version du 20 mai 2012

Table des matières

Table des figures

Introduction

Dans le cadre de notre formation au sein du Département Informatique de l'école Polytechnique de l'Université de Tours, un projet de science de la décision est effectué afin de mettre en pratique nos connaissances acquises au cours de notre formation.

Le but de ce projet est d'effectuer un pré-traitement de données publiques (Open Data) qui est composé d'un chargement, d'un nettoyage et d'une reconnaissance. La partie chargement ayant déjà été effectuée par un étudiant de 5ème année, notre encadrant nous a donc amené à laisser la partie chargement de côté et de se charger des deux autres parties citées précédemment.

Nous tenons à remercier notre encadrant Mr. Gilles Venturini pour ses conseils pertinents qui nous ont été d'un appui considérable pour accomplir notre projet.

Notre rapport est constitué essentiellement de quatre parties. La première partie définit le cahier des charges. La deuxième partie est consacrée aux spécifications du système. La troisième partie explique la conception de l'application. Et enfin la quatrième partie présente l'implémentation de l'application.

1. Partie 1 : Cahier des Charges

1.1 Contexte

Data.gouv.fr est la plateforme française d'ouverture des données publiques (« Open Data »). Ce portail unique interministériel est destiné à rassembler et à mettre à disposition librement l'ensemble des informations publiques de l'Etat, de ses établissements publics administratifs et, si elles le souhaitent, des collectivités territoriales et des personnes de droit public ou de droit privé chargées d'une mission de service public.

Les données publiques recensées sur data.gouv.fr sont réutilisables librement et gratuitement, dans les conditions définies par la « Licence Ouverte / Open Licence »

Même si ces données sont libres et consultables par tous, il semble intéressant de créer un système permettant d'accéder à ces données , mais aussi de créer une cartographie (objet à deux dimensions).

C'est de cette idée que vient le principe même de ce projet.

1.2 Présentation du projet

1.2.1 Présentation générale

Ce projet a été découpé en plusieurs parties qui ont été réparties dans 3 groupes différents. La partie qui nous a été affectée est la deuxième :

- La première partie consiste à télécharger automatiquement ces données, les nettoyer et reconnaître automatiquement les variables présentes et leurs types, puis proposer un format de sortie commun (XML).
- La seconde partie (qui est celle qui nous concerne) consiste à analyser ces fichiers XML et à générer une matrice.
- La dernière partie consiste à mesurer les distances entre fichiers et à créer une cartographie de ces fichiers.

La partie téléchargement des données étant déjà effectuée, nous travaillons donc avec un ensemble de dossiers contenant des fichiers au format XML et Excel (.XLS). A noter que pour chaque fichier XLS correspond un fichier XML. Chaque paire (XML et XLS) de fichiers correspond à une donnée publique.

Voici comment cela se passe lorsque l'on tente d'accéder à une donnée publique présente sur le site web http ://www.data.gouv.fr/. Les numéros indiquent l'ordre dans lequel se font les différentes interactions.

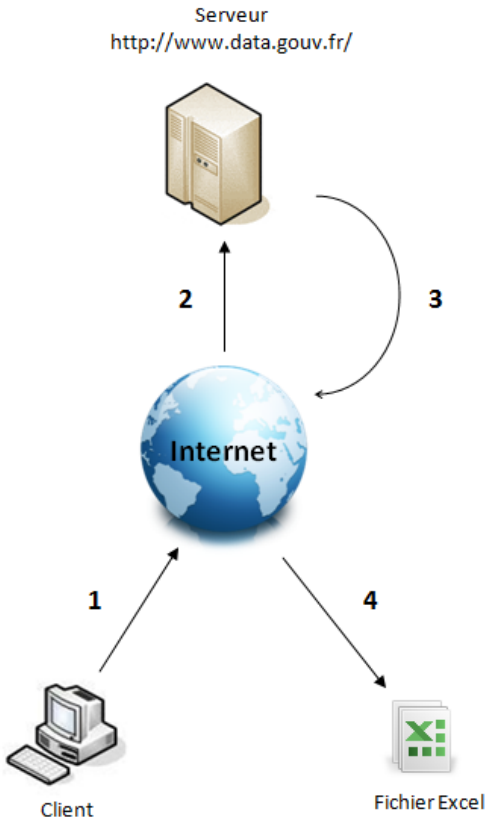

FIGURE 1.1 – Fonctionnement téléchargement donnée publique

Cependant, nous nous intéresserons qu'aux fichiers XML. Ces derniers sont constitués de la façon suivante.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata Filename="nom_fichier.xls">
    <DownloadLink></DownloadLink>
    <Title></Title>
    <Publisher></Publisher>
    <DateTime></DateTime>
    <Description></Description>
</metadata>
```

FIGURE 1.2 – Structure fichier XML

Les différents balises qui constitue chaque fichier XML sont :

- Metadata : contient le nom du fichier XLS auquel est rattaché ce fichier XML
- DownloadLink : lien internet qui a permit de télécharger le fichier XLS
- Title : titre de la donnée publique
- Publisher : le responsable de la publication de cette donnée
- DateTime : la date de publication de cette donnée
- Description : description de la donnée publique

Afin de mieux visualiser à quoi peut ressembler un fichier XML complet, voici un exemple de ce type de fichier.

```xml
<?xml version="1.0" encoding="utf-8"?>
<metadata Filename="EXPCC_FAM_COM2B185_08.xls">
  <DownloadLink>http://www.recensement-2008.insee.fr/exportXLSCC.action?idTheme=4&amp;codeZone=2B185-COM</DownloadLink>
  <Title>Recensement de la population, tableau de chiffres clés - Couples - Familles - Ménages : Oletta (2B185 - Commune)</Title>
  <Publisher>Institut national de la statistique et des études économiques (INSEE)</Publisher>
  <DateTime>10-01-2012</DateTime>
  <Description>Ce jeu de données propose pour la zone géographique concernée un tableau synthétique d'indicateurs issus du Recensement
  de la population sur le domaine suivant : Couples - Familles - Ménages. D'autres jeux de données (un jeu de données par zone géographique)
  proposent le même tableau pour divers niveaux géographiques du niveau communal au niveau national. La base de données fournissant les
  valeurs de ces indicateurs pour chacune des communes de France et arrondissements municipaux de Paris, Lyon et Marseille est également
  disponible dans un jeu de données. Statistique publique.</Description>
</metadata>
```

FIGURE 1.3 – Exemple fichier XML

1.2.2 Présentation de notre partie

Notre projet consiste donc à analyser des fichiers XML qui ont été préalablement générés.

Notre application doit :

- Ouvrir chaque fichier XML
- Récupérer le texte, séparer et nettoyer (enlever caractères spéciaux...) chaque mot
- Faire le tri entre mot à conserver ou à rejeter
- Tronquer les mots au deuxième bloc de voyelle rencontré (**or**d**in**ateur : ordi ; b**ouill**ir : bouilli ...)
- Énumérer chaque mot trouvé
- Retourner une matrice qui renseigne sur le nombre de mot répertorié dans chaque fichier (ainsi que d'autres informations comme le nombre de mots conservés etc...)

La matrice retournée aura donc cette forme :

Fichiers/Variable	var1	var2	var3	var4	var5
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

FIGURE 1.4 – Forme matrice

L'application doit également proposer à l'utilisateur de sélectionner les balises (Title, DateTime...) qu'il souhaite analyser.

1.3 Contraintes

1.3.1 Contraintes de délais

- Début du projet : Vendredi 9 Mars
- Fin du projet : Lundi 21 Mai

La durée totale du projet est donc d'environ 2 mois et demi.

1.3.2 Contraintes techniques

La contrainte technique la plus importante est que le temps de la réalisation de la tâche doit rester "raisonnable".

En effet, le nombre de fichiers se comptant en milliers, les traitements doivent prendre le moins de ressources possible.

1.3.3 Contraintes de réalisation

Nous n'avons pas de contraintes de réalisations spécifiques. En effet, le choix du langage et des différents outils de programmation reste totalement libre.

1.4 Système de gestion de versions

Nous utilisons un système de gestion de versions pour travailler plus efficacement durant ce projet. Ceci ne nous était pas imposé, mais il nous a semblé judicieux d'en utiliser un, pour gagner en temps et en efficacité.

FIGURE 1.5 – Logo Subversion

Cela nous permet durant tout le projet de :

- Pouvoir revenir en arrière
- Voir qui a modifié ce fichier en dernier

Nous avons choisi Subversion (SVN). Ce choix s'explique par plusieurs critères :

- Documentations très riches, forums actifs

– Interfaces graphiques sous Windows : intégré à l'explorateur via le plugin TortoiseSVN Nous avons également décidé d'héberger notre espace de travail sur le site Assembla. Cela nous a permis de pouvoir transférer et récupérer notre travail de partout (Université, domicile...)

2. Partie 2 : Spécifications du système

2.1 Introduction

Ce projet a pour but de concevoir et de développer une application dans le langage de notre choix (C++ préconisé) qui traite un grand nombre de fichiers XML.

2.2 Contexte de la réalisation

Ce projet entre dans le contexte du projet de 4ème année intitulé "Science de la décision" et a pour objectif de mettre en œuvre les principaux concepts vus en cours.

2.3 Fonctions générales du système

Les fonctions générales du système sont visibles dans la partie 1.2.2 Présentation de notre partie.

2.4 Interface utilisateur

2.4.1 Interface de la fenêtre principale

Au démarrage de l'application, l'utilisateur peut voir cette fenêtre :

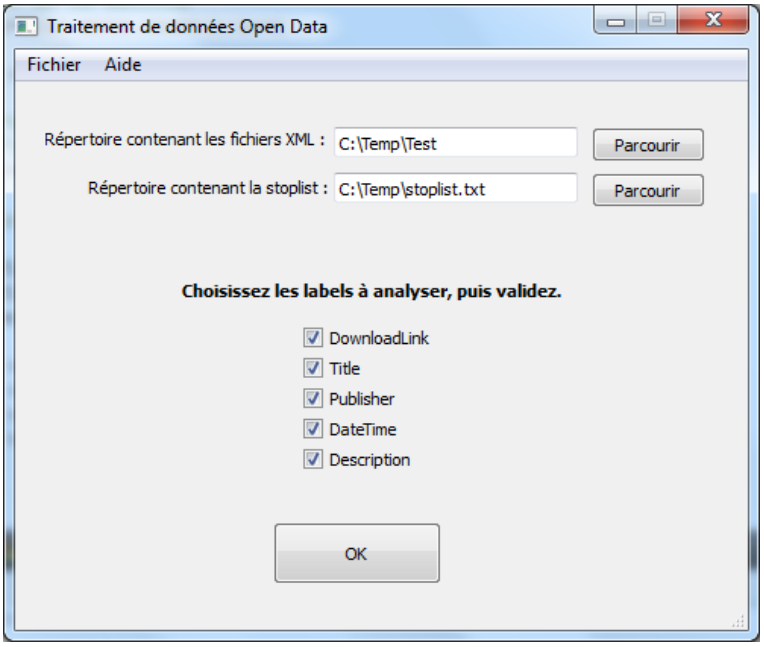

FIGURE 2.1 – Fenêtre principale

L'utilisateur peut choisir le répertoire contenant les fichiers XML ainsi que le répertoire contenant la stoplist. De plus, il a la possibilité de choisir les labels à analyser.

2.5 Description de l'environnement

Le système (application Open Data) fonctionne sous Windows et interagit avec un acteur humain. On peut représenter l'environnement du système comme ceci :

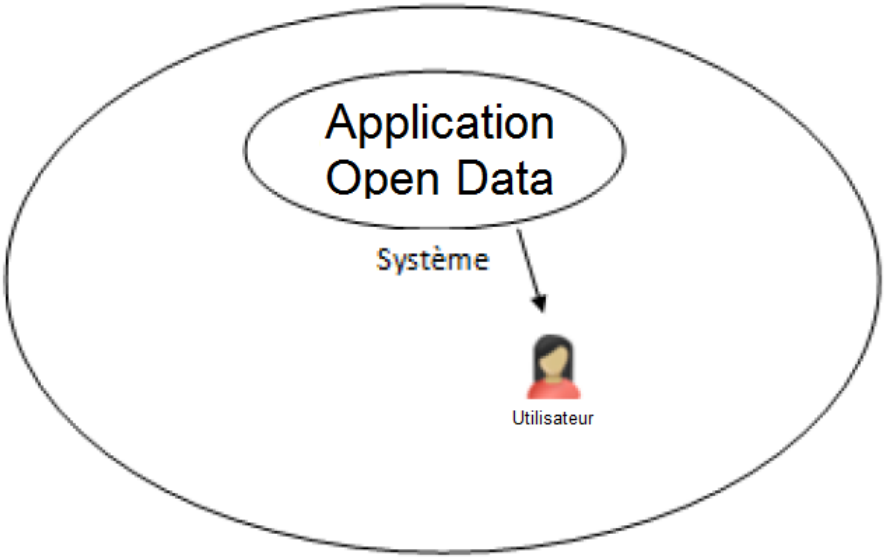

FIGURE 2.2 – Schéma environnement système

2.6 Description des objets

On définit les objets échangés entre le système et l'environnement en construisant l'arbre hiérarchique des objets ci-dessous :

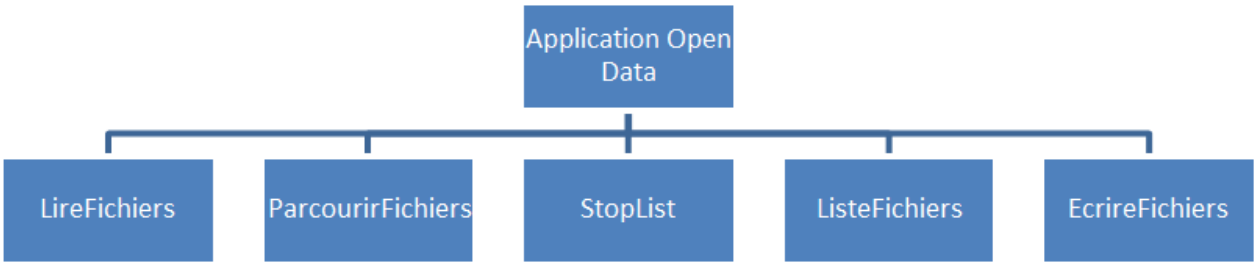

FIGURE 2.3 – Arbre hiérarchique des objets

2.7 Description des fonctionnalités

Après la modélisation, on définit les fonctionnalités proposées par le système avec l'arbre hiérarchique des fonctionnalités.

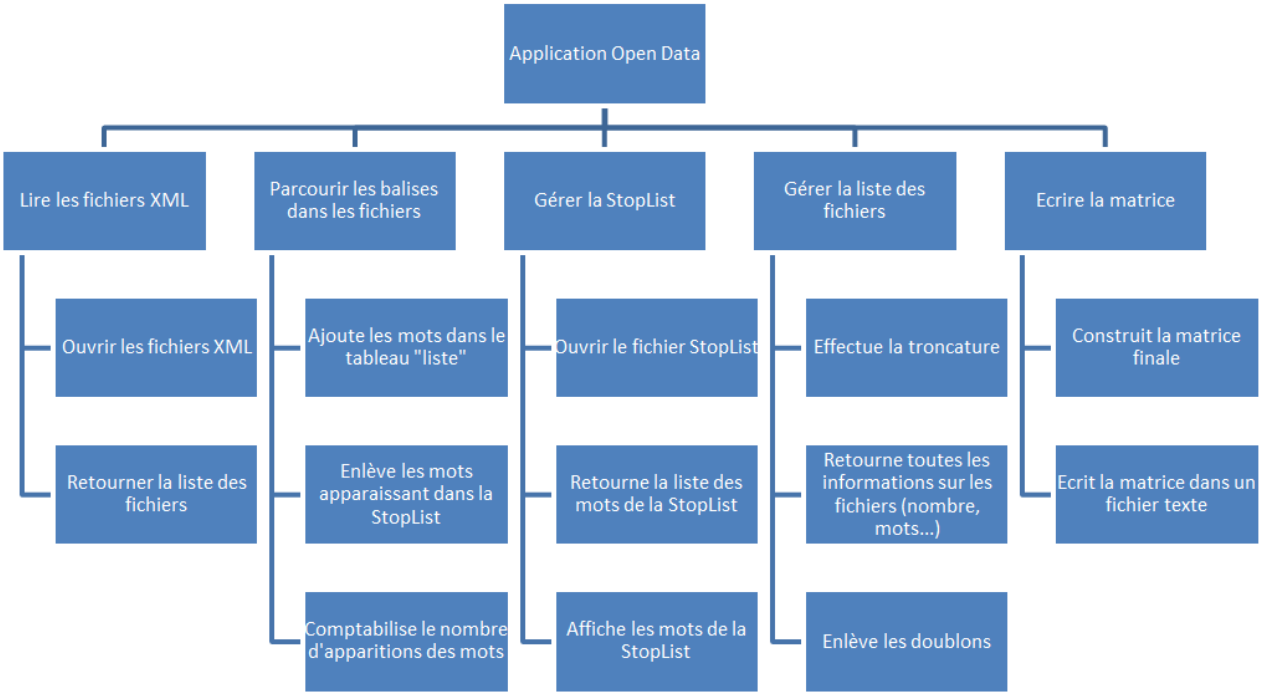

FIGURE 2.4 – Arbre hiérarchique des fonctionnalités

3. Partie 3 : Conception de l'application

3.1 Solution adoptée

3.1.1 Problématique

Notre principale problématique a été de savoir comment stocker les différents mots ainsi que le nombre d'apparitions de ces mots.

Sachant que nous utilisions le Framework Qt, différentes solutions intéressantes s'offraient à nous.

Parmi ces choix, les plus intéressant au niveau des traitements que nous devons effectuer sur ces listes, sont QVector et QList.

3.1.2 Solution choisie

Après de nombreuses lectures sur la documentation officielle de Qt ainsi que sur des forums spécialisés, nous avons trouvé que QList était plus rapide que QVector.

Comme le nombre de traitement est assez important, et que le nombre de fichier l'est encore plus, il nous a parut logique d'utiliser les QList.

Première étape

La classe ListeFichiers comporte ces deux variables :

- QList<QStringList> liste ;
- QList<QList<int> > nbApparitions ;

Le premier élément "liste" est un tableau à 2 dimensions (un QList de QStringList). Chaque ligne représente un fichier XML. Chaque colonne représente un mot contenu dans le fichier.

Par exemple, dans le tableau ci dessous, chaque lettre représente un mot différent. Le premier fichier comporte donc 4 mots qui sont E, F, G et B. Le deuxième fichier comporte 5 mots qui sont B, E, H, I et C.

Il ne peut pas y avoir de doublons dans une même ligne, par contre, il peut y avoir des mots identiques dans différents fichiers (E est présent dans le fichier 1 et dans le fichier 2)

1	E	F	G	B		
2	B	E	H	I	C	
3	L	M	N	A	J	K
4	A	C	Z	O		
5	F	R	J	I	P	
6	D	F	I	A	E	

FIGURE 3.1 – Explication tableau "liste"

Le deuxième élément "nbApparitions" est un tableau à 2 dimensions (un QList de QList).

Il aura forcément les mêmes dimensions que le tableau "liste".

Par exemple, dans le tableau ci-dessous, on peut voir que dans le premier fichier, le mot E apparaît deux fois, le mot F apparaît une fois, le mot G apparaît 3 fois...

1	2	1	3	1		
2	2	1	1	1	1	
3	2	2	5	6	1	2
4	5	4	1	2		
5	4	2	1	1	2	
6	1	2	3	5	5	

FIGURE 3.2 – Explication tableau "nbApparitions"

Deuxième étape

Ces deux tableaux vont nous permettre de créer la matrice.

Pour cela, dans la classe EcrireFichier, nous avons déclaré deux variables :

- QStringList mots ;
- QList<QList<int> > matrice ;

La première variable (mots) est simplement un tableau de String à une dimension. Il contient tous les mots tronqués de tous les fichiers. Ils sont présents une et une seule fois.

E	F	G	B	H	I	C	L	M	N	A	J	K	Z	O	F	R	P	D

FIGURE 3.3 – Explication "mots"

La deuxième variable (matrice) est un tableau d'entiers à deux dimensions. Ce tableau contient les itérations des différents mots. La matrice a cette forme :

1	2	1	3	1															
2	1			2	1	1	1												
3								2	2	5	6	1	2						
4							4				5			1	2				
5						1						1				4	2	2	
6	5					3					5					2			1

FIGURE 3.4 – Explication matrice

Afin de rendre l'illustration de cette matrice plus claire, les 0 n'apparaissent pas ici. En effet, la matrice ne contient pas de "vide", mais des 0 lorsque le nombre d'apparitions est nul.

De nombreux traitements sont effectués pour obtenir les deux listes de la première étape, et pour passer à la deuxième étape.

3.2 Déroulement de l'application

Dans cette partie, nous allons tenter d'expliquer comment se déroule l'application. Pour cela nous allons expliquer les différentes étapes qui sont effectuées lorsque l'utilisateur lance le traitement.

Une fois le chemin du répertoire contenant les fichiers XML, le chemin du fichier contenant la stoplist et les labels à analyser choisis (au moins un doit être choisi), le traitement peut démarrer.

3.2.1 Lecture des fichiers

Premièrement nous faisons appel à :

LireFichier *lirefic = new LireFichier() ;

Cela va permettre de lister les fichiers XML. Pour cela, nous avons utilisé QtXml. Il s'agit du module qui permet d'utiliser des données XML dans un programme Qt.

3.2.2 Parcours des fichiers

Ensuite, on fait appel à :

new ParcourirFichier(listeDesFichiers, lirefic->getFileList()) ;

Cette classe permet de mettre les mots dans le tableau "liste", de compter les itérations et de les placer dans le tableau "nbApparitions". (cf. 3.1.2. Solution choisie)

3.2.3 Écriture des fichiers

Enfin, on fait appel à :

new EcrireFichier(listeDesFichiers) ;

Cette classe permet d'abord de créer la matrice et le tableau de mots, puis d'écrire la matrice dans le fichier texte.

3.2.4 Schéma illustrant ce déroulement

Afin de compléter l'explication sur le déroulement de l'application. Ce schéma permet d'illustrer ce déroulement.

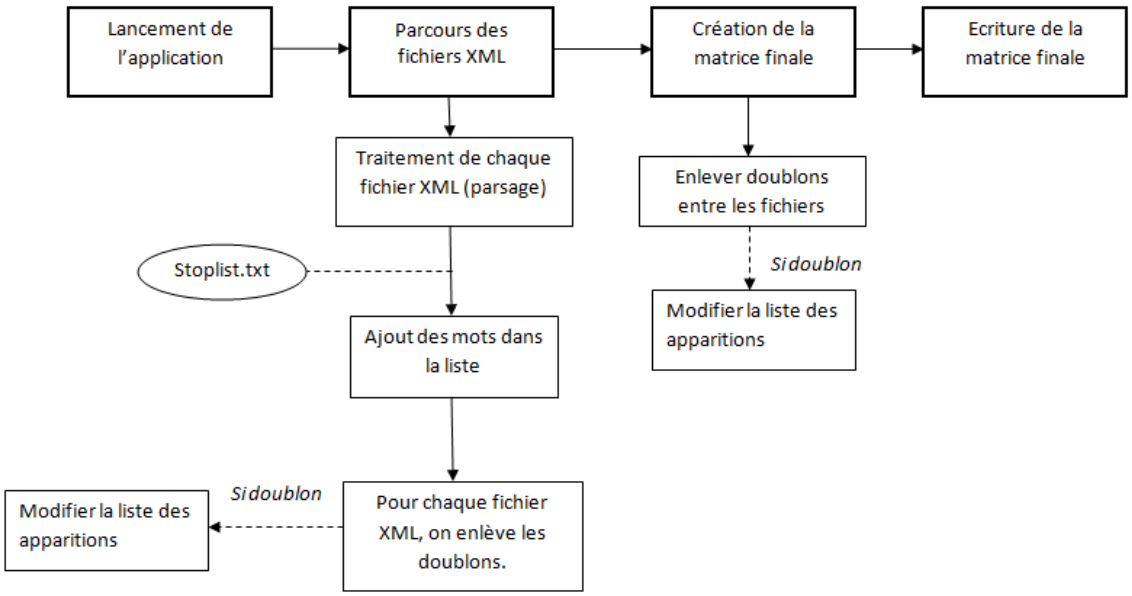

FIGURE 3.5 – Déroulement application

La stoplist est un fichier texte contenant une liste de mots à ne pas prendre en compte. On peut par exemple citer « aussi », ou « auquel », qui sont des mots qui ne nous intéressent pas.

3.3 Explication du code

3.3.1 Utilisation des expressions régulières

Nous avons utilisé des expression régulières pour répondre à trois besoins :

- L'application effectue une troncature des mots au deuxième bloc de voyelles non consécutifs.
- Notre application devant traiter des mots pendant tout le traitement, nous avons "nettoyé" ces mots. En effet, beaucoup des mots commencent ou finissent par des caractères spéciaux et certains en contiennent (apostrophe).
- Certains mots sont écartés du traitement car jugés non utiles.

Troncature

Pour éviter de traiter un trop grand nombre de mots et avoir en sortie une matrice trop grosse, il a été demandé de tronquer les mots.

Cette troncature ne s'effectue pas aléatoirement ou après un nombre de caractère donné, mais après deux voyelles non consécutives.

Par exemple :

- bouillir -> bouilli

– président -> prési
– eau -> eau

Le code est visible ci-dessous :

```
QString ListeFichiers::troncature(QString mot)
{
    int longueur;
    QRegExp eReg("^[^âàaeéèêiouy]*[âàaeéèêiouy]+[^âàaeéèêiouy]+[âàaeéèêiouy]",Qt::CaseInsensitive);

    int debut = eReg.indexIn(mot);

    if (eReg.capturedTexts().at(0)!="")
    {
        longueur = eReg.capturedTexts().at(0).count();
        return mot.mid(debut,longueur);
    }
    return mot;
}
```

FIGURE 3.6 – Procédure troncature

L'expression régulière indique que le mot commence par 0 à plusieurs consonnes, puis au moins une voyelle, suivi d'au moins une consonne, puis une voyelle.

Si le mot est dans ce cas, ou coupe le mot après la 2ème voyelle non consécutive.

Nettoyage des mots

Pour nettoyer les mots, nous avons défini les différentes possibilités de placement des caractères spéciaux.

Ainsi, si trois caractères spéciaux sont présents à la fin du mot (... par exemple), le mot est coupé avant ces trois caractères.

```
QString ListeFichiers::motNettoye(QString mot)
{
    // Si le mot finit par trois caractères spéciaux
    QRegExp eReg3("[^\\w]{3}$",Qt::CaseInsensitive);
    if (mot.contains(eReg3))
    {
        mot.resize(mot.size()-3);
    }

    // Si le mot finit par un caractère spécial
    QRegExp eReg2("[^\\w]{1}$",Qt::CaseInsensitive);
    if (mot.contains(eReg2))
    {
        mot.resize(mot.size()-1);
    }

    // Si le mot commence par un caractère spécial
    QRegExp eReg1("^[^\\w]{1}",Qt::CaseInsensitive);
    if (mot.contains(eReg1))
    {
        mot.remove(0,1);
    }

    // Si le mot conteitn un caractère alphabétique, puis un caractère spécial
    QRegExp eReg("^[\\w]{1}[^\\w]{1}",Qt::CaseInsensitive);
    if (mot.contains(eReg))
    {
        mot.remove(0,2);
    }

    return mot;
}
```

FIGURE 3.7 – Procédure motNettoye

Mots enlevés

Enfin, nous avons défini des conditions qui permettent de dire si oui ou non le mot doit être pris en compte (en plus de la stopList).

```
bool ListeFichiers::nePasMettre(QString mot)
{
    // Contient des chiffres
    QRegExp eReg1("[0-9]+",Qt::CaseInsensitive);

    // Ne contient qu'un seul caractère
    QRegExp eReg2("^.{1}$",Qt::CaseInsensitive);

    // Ne contient que 2 caractères
    QRegExp eReg3("^.{2}$",Qt::CaseInsensitive);

    // Commence par http
    QRegExp eReg4("^http",Qt::CaseInsensitive);

    // Est un caractère spécial (ou plusieurs)
    QRegExp eReg5("^[^\\w]+$",Qt::CaseInsensitive);


    if (mot.contains(eReg1) || mot.contains(eReg2) || mot.contains(eReg3) || mot.contains(eReg4) || mot.contains(e
        return true;

    return false;
}
```

FIGURE 3.8 – Procédure nePasMettre

Les mots que nous souhaitons ne pas traiter sont ceux contenant des chiffres (année, numéro ...), les mots contenant un seul ou deux caractères, les adresses internet ainsi que les caractères spéciaux (:).

4. Partie 4 : Implémentation de l'application

4.1 Langage et outils utilisés

4.1.1 Langage C++

Nous avons décidé de programme cette application en C++ pour de multiples raisons.

- La première est que ce langage est assez rapide et que cela est un critère important dans ce projet (où l'on traite des milliers de fichiers).
- Ce langage de programmation permet la programmation orientée objet.
- C++ est actuellement un des langages les plus utilisés.
- Nous avons appris à utiliser le Framework Qt cette année et nous voulions réaliser un projet avec ce dernier.

Framework Qt

Qt est un framework orienté objet et développé en C++. Il offre des composants d'interface graphique (widgets), d'accès aux données, d'analyse XML...

Utiliser Qt était le plus judicieux pour nous, de par la richesse de son API (Application Programming Interface).

FIGURE 4.1 – Logo Qt

4.1.2 Outils utilisés

Qt Creator

Qt Creator est un environnement de développement intégré multiplate-forme faisant partie du framework Qt. Il est donc orienté pour la programmation en C++. Il intègre directement dans l'interface un débogueur, un outil de création d'interfaces graphiques ainsi que la documentation Qt.

L'éditeur de texte intégré permet l'autocomplétion ainsi que la coloration syntaxique.

Nous avons utilisé Qt Creator sous Windows avec le compilateur MinGW.

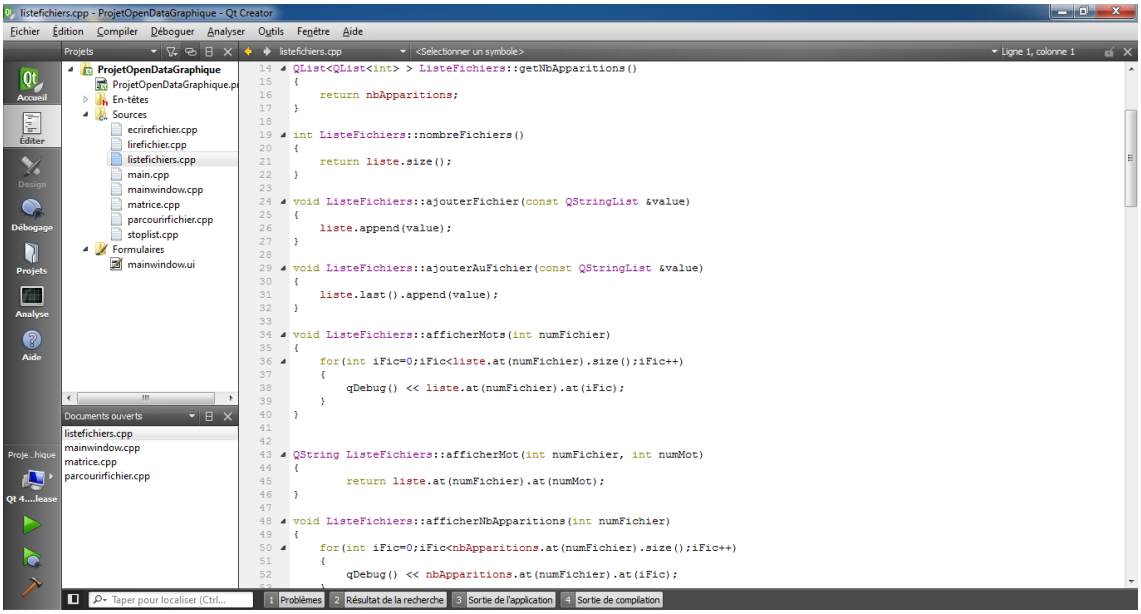

FIGURE 4.2 – Qt Creator

4.2 Problèmes rencontrés

Nous avons rencontrés, pendant le projet, plusieurs problèmes, ce qui est tout à fait normal.

4.2.1 Caractères spéciaux

La liste des mots après nettoyage et troncature contenait soit des caractères spéciaux (' ; " ...) soit des espaces vides.

Pourtant, nous effectuons une vérification. En effet, si le mot contient moins de 3 caractères, il est éliminé. Nous avons résolus ces problèmes avec l'utilisation des expressions régulières.

4.2.2 Temps de traitement

Malgré tous nos efforts pour essayer d'optimiser les fonctions, nous observons des temps de traitement très long pour un grand nombre de fichiers (>1000).

Nous savons quels sont les traitements les plus long, et comprenons que cela prenne du temps. En effet, dans la fonction enleveDoublonsFichiers(), chaque mot est comparé à tous les autres (3 for

imbriqués). Ce problème est toujours d'actualité, et nous n'avons pas eu le temps nécessaire pour réduire d'avantage ce temps d'exécution.

Cependant, nous avons réussi à gagner presque 2 heures de traitement sur un nombre de fichiers de 500 (cf . **4.3 Performances de l'application**) en remplaçant les crochets ([]) par des at(). Nous avons découvert que at() était plus rapide en lisant la documentation officielle de Qt.

4.3 Performances de l'application

Comme expliqué dans la section **4.2 Problèmes rencontrés**, il est nécessaire de comparer chaque mot à tous les autres pour voir si ce dernier est déjà présent.

Cela devient gênant avec un très grand nombre de fichiers car le temps de traitement est de plus en plus important.

On peut voir dans le tableau ci-dessous les performances obtenues sur quelques tests.

Nombre de fichiers	Temps de traitement (en s)	Taille approximative Matrice
1	<1	1 * 35
10	<1	10 * 118
100	32	100 * 169
500	822 = 13min42s	500 * 646
1000	2864 = 47min44s	1000 * 1237
2000	12280 = 3h24min40s	2000 * 1754

FIGURE 4.3 – Performances

5. Conclusion

Ce projet nous a été bénéfique sur différents points, puisqu'il nous a permis de travailler en binôme, de créer des algorithmes optimisés, et de travailler pour la première fois sur un vrai projet en C++.

Cela nous a aussi permis de mieux maitriser le framework Qt ainsi que l'environnement de développement QtCreator. En effet, nous avons déjà eu l'occasion de travailler avec ce framework cette année. Cependant, une grande partie du travail concernait la partie graphique de l'application, contrairement à ce projet, où, la partie traitement était la plus importante.

Nous nous sommes partagés les tâches et nous nous sommes concertés en permanence pour nous assurer du bon déroulement du projet. En effet, la réalisation d'un projet dans un temps déterminé et avec un but précis oblige à devoir s'organiser et planifier toutes les étapes de la réalisation.

Malgré quelques contraintes (techniques et de réalisations), ce projet s'est avéré très intéressant et enrichissant. Le travail en binôme nous a permis de nous exercer sur le travail en équipe, ce qui est très important dans notre future vie professionnelle.

Annexes

Exemple de fichier XML traité

```xml
<?xml version="1.0" encoding="utf-8"?>
<metadata Filename="EXPCC_FAM_COM24122_08.xls">
  <DownloadLink>http://www.recensement-2008.insee.fr/exportXLSCC.action?idTheme=4&amp;codeZone=24122-COM</DownloadLink>
  <Title>Recensement de la population, tableau de chiffres clés - Couples - Familles - Ménages : Cladech (24122 - Commune)</Title>
  <Publisher>Institut national de la statistique et des études économiques (INSEE)</Publisher>
  <DateTime>10-01-2012</DateTime>
  <Description>Ce jeu de données propose pour la zone géographique concernée un tableau synthétique d'indicateurs issus du Recensement
  de la population sur le domaine suivant : Couples - Familles - Ménages. D'autres jeux de données (un jeu de données par zone géographique)
  proposent le même tableau pour divers niveaux géographiques du niveau communal au niveau national. La base de données fournissant les
  valeurs de ces indicateurs pour chacune des communes de France et arrondissements municipaux de Paris, Lyon et Marseille est également
  disponible dans un jeu de données. Statistique publique.</Description>
</metadata>
```

FIGURE 1 – Exemple fichier XML

Matrice résultante

```
1 Rece;popu;table;chiffre;clés;Couple;Fami;Ména;Clade;Commu;Insti;nati;stati;étu;éco;
  INSE;jeu;donné;propo;zone;géogra;conce;synthé;indi;issu;doma;jeux;nive;base;fourni;
  vale;France;arro;muni;Pari;Lyon;Marse;éga;dispo;publi;nbMots;nbApparitions
2 2;2;3;1;1;2;2;2;1;3;1;2;2;1;1;1;3;5;2;2;3;1;1;2;1;1;1;3;1;1;1;1;1;1;1;1;1;1;1;1;40;64
```

FIGURE 2 – Matrice résultante

Pré-traitement de données publiques Open Data

Département Informatique
4e année
2011 - 2012

Rapport de projet Science de la Décision

Résumé : Application en C++ (Qt) de lecture de fichiers XML et de traitement (nettoyage, troncature...) des mots présents. Retourne une matrice (fichier/variable) dans un fichier texte.

Mots clefs : Open Data ; pré-traitement ; données publiques ; gouvernement

Abstract: Application in C + + (Qt) for reading XML files and words processing (cleaning, ... truncation). Returns a matrix (file / variable) in a text file.

Keywords: Open Data; pretreatment ; public data ; government

Encadrants
Gilles VENTURINI
gilles.venturini@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Rémi CARUYER
remi.caruyer@etu.univ-tours.fr
Pierre FOURREAU
pierre.fourreau@etu.univ-tours.fr

DI4 2011 - 2012