2014
2015

Rapport de stage

- Développement d'un outil d'analyse de poèmes -

De la poésie des chiffres aux chiffres de la poésie…

Réalisé au sein du Centre Informatique de Philosophie et Lettres de l'Université de Liège dans le cadre du stage de première année de Master informatique

RINGWALD Célian
Encadré par Laurent SIMON (CIPL)
& Julien AH-PINE (Université Lyon 2)

UNIVERSITÉ LUMIÈRE LYON 2
UNIVERSITÉ DE LYON

icom
INSTITUT DE LA COMMUNICATION

Département d'Informatique et de Statistique
UNIVERSITÉ LUMIÈRE LYON 2
DIS
Institut de la Communication

RESUME

Ce rapport présente le travail réalisé durant mon stage de fin d'année d'étude de master 1 informatique au centre informatique de philosophie et lettre (CIPL) à Liège. La mission de celui-ci consistait à réaliser un programme permettant de comparer une suite de mots, un poème ou un recueil de poèmes afin de les comparer avec un corpus donné. Ce logiciel, Poestry Miner, présenté dans sa réalisation et sa forme finale, permet donc d'obtenir dans ce cadre une liste de poèmes résultats ainsi que des indicateurs permettant à un linguiste de mesurer le degré de pertinence de ceux-ci.

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont contribuées au succès de mon stage, qui m'ont aidées lors de la rédaction de ce rapport, et m'ont beaucoup apportées.

Tout d'abord, j'adresse mes remerciements à Magali, papyrologue, qui au cours d'un covoiturage m'a permis de faire la connaissance du LASLA et du CIPL, ainsi qu'à Monsieur Longree qui m'a rapidement répondu lors de mon premier mail et dirigé vers Monsieur Purnelle.

J'adresse de plus, un grand merci à Monsieur Purnelle, qui m'a offert la possibilité de choisir mon sujet et m'a permis de m'épanouir dans ma mission. Je le remercie aussi d'avoir su libérer de son temps lorsque j'ai eu des difficultés à concevoir certains points de la demande. Et qui a su, indirectement me donner goût à la poésie.

Je remercie, bien sûr, Monsieur Simon qui fût mon tuteur et qui m'a fait une place dans son bureau. Ce fût de plus un plaisir d'échanger au quotidien à ses côtés, toujours disponible et à l'écoute pour répondre à mes nombreuses interrogations.

Je tenais à remercier aussi toute l'équipe du CIPL, qui m'a rapidement intégrée et avec qui ce fût un plaisir de partager une tarte le jeudi. Je remercie plus particulièrement M. Putz de m'avoir fait partager sa passion pour la photographie, et grâce à qui je me suis dès mon retour en France acheté un appareil argentique. Et bien entendu M. Dupuy qui m'aura permis de goûter de délicieuses gaufres belges.

Enfin je remercie la région Rhône-Alpes qui m'a permis de financer ce stage non rémunéré via la bourse Explo'RA Sup.

SOMMAIRE

1. INTRODUCTION 4

2. CONTEXTE 5

2.1. LE LASLA 5

2.2. LE CENTRE INFORMATIQUE DE PHILOSOPHIE ET LETTRES 5

2.3. PRESENTATION DE LA MISSION 6

3. LA GESTION DU PROJET 8

3.1. CAHIER DES CHARGES 8

3.2. ANALYSE DE L'EXISTANT 9

3.3. CHOIX DES TECHNOLOGIES 10

3.4. DEROULEMENT DE LA MISSION 10

4. MISE EN OEUVRE DU PROGRAMME 11

4.1. CONCEPTION 11

4.1.1. Stratégie globale de la démarche 11

4.1.2. Choix de la structure des fichiers poèmes et recueils 11

4.1.3. Conception de la base de données 12

4.1.4. Le modèle Entité-Objet 12

4.1.5. Stratégies de réduction de la liste de résultats 13

4.1.6. Classes liées aux traitements des requêtes 14

4.1.7. Modélisation de l'interface 15

4.2. IMPLEMENTATION 16

4.2.1. Gestion de la base de données 16

4.2.2. Anti-dictionnaire 17

4.2.3. Le calcul des résultats 18

4.2.4. Mise en forme des poèmes 18

4.2.5. Export et création de rapports 19

4.3. DEBOGAGE, TEST ET OPTIMISATION 19

4.3.1. Débogage 20

4.3.2. L'optimisation 20

4.4. VALIDATION 22

4.5. REDACTION DE LA DOCUMENTATION 22

5. LE RESULTAT : POESTRY MINER 23

5.1. PRESENTATION DU PROGRAMME 23

5.1.1. Interface principale 23

5.1.2. Interface de résultats 24

5.1.3. Interface de gestion de corpus 25

5.2. PERFORMANCES ET LIMITES 26

5.2.1. Export de fichier 26

5.2.2. Import de fichier 26

5.2.3. Requêtes de type recueils 27

5.2.4. Requêtes de type poèmes 28

5.2.5. Requêtes de type texte 29

5.3. POUR ALLER PLUS LOIN 30

6. CONCLUSION 31

1. INTRODUCTION

Après avoir réalisé un stage en informatique décisionnelle pour une entreprise privé, ainsi qu'un second dans un observatoire publique, j'ai choisi de m'approcher cette année du monde universitaire en réalisant mon stage au Centre Informatique de Philosophie et Lettre de l'Université de Liège. Les mathématiques et l'informatique sont aujourd'hui présents dans de nombreux domaines d'activité, ce qui nous permet en tant qu'étudiants statisticiens de disposer de l'embarrât du choix quant à notre futur environnement de travail, c'est en ça que ces stages sont intéressants, en plus d'être très pédagogique.

Curieux de comprendre comment il était possible de quantifier des mots je me suis donc présenté en début d'année auprès de M. Purnelle philologue et membre du Laboratoire d'Analyse Statistique de Langues Ancienne, afin de savoir si mission pouvait m'être confiée. C'est ainsi que j'ai pu être intégré durant ces trois mois très enrichissants en tant que stagiaire dans les locaux du CIPL.

En possession de nombreux manuscrits d'illustres poètes Liégeois (Izoard et Jacqmin notamment), M. Purnelle a pour mission d'éditer l'œuvre de ceux-ci afin de mettre à jour ce patrimoine culturel encore enfoui entre les lignes. C'est en éditant ces poèmes inédits et oubliés, que ce spécialiste permet de donner ces lettres de noblesses à ces deux poètes encore trop méconnus du grand public. Dans cette optique, ma mission consistait à développer un outil logiciel permettant aux spécialistes de découvrir ces pépites ; un stage fort enrichissant pour moi sur le plan culturel, technique et professionnel.

N'étant jusqu'à présent que peu à l'aise avec le développement logiciel, j'ai pu par ce biais me perfectionner. Conscient de ce réel manque dans ma boite à outils de compétences, j'ai pu découvrir ce que certains appellent « les joies de la programmation ». Une expérience très bénéfique selon moi qui m'obligeât à me retrousser les manches. Me revendiquant jusqu'ici statisticien de par ma formation, je ne pouvais faire l'impasse sur cette compétence que l'on attend du scientifique de la donnée.

Ce rapport est composé de cinq parties, la première inscrit mon stage dans son contexte, en présentant le centre, le travail de mon maître d'ouvrage, les poètes étudiés. La seconde partie expose la face gestion de projet de mon stage, en exprimant le contenu du cahier des charges, les choix techniques envisagés ainsi que le planning et la méthode de travail mise en oeuvre. Suite à cela, je dresserai dans la troisième partie la mise en œuvre du logiciel : de sa conception à son implémentation. La dernière partie, dresserai l'état des lieux de ce qui a été réalisé : le programme, ses performances et ses limites, ce qui aurait pu être fait pour améliorer celui-ci. Enfin, je conclurai en évoquant les perspectives de ma mission au sein du centre, ainsi que l'impact de cette expérience sur mon projet professionnel. La problématique de mon stage étant de comprendre comment passer de la poésie des chiffres aux chiffres de la poésie, j'espère que ce dossier saura vous éclairer sur le sujet d'une agréable manière.

Bonne lecture,

2. CONTEXTE

Photographie de l'Université de Philosophie et Lettres de Liège @Brunot Devoghel

2.1. LE LASLA

Le Laboratoire d'Analyse Statistique des Langues Anciennes (LASLA) est né à l'initiative d'Etienne Evrard en collaboration avec Louis Delatte en 1961. Ce laboratoire fût l'un des précurseurs dans le domaine de l'analyse des données textuelles et de la littérature en utilisant l'informatique et la statistique comme outils de travaux (rappelons au passage que l'ordinateur d'aujourd'hui est bien différent de celui d'antan, en effet celui-ci était encore mécanique et les enregistrements se faisaient via des cartes perforées). Les travaux de ce laboratoire dans l'étude en langues greco-romanes, et dans la constitution de bandes de données littéraires en font aujourd'hui un des acteurs majeur de son domaine.

2.2. LE CENTRE INFORMATIQUE DE PHILOSOPHIE ET LETTRES

Le CIPL (Centre Informatique de Philosophie et lettres) est né du développement de l'informatique dans la faculté sous l'impulsion des membres du LASLA. Il a pour objectif de promouvoir et coordonner l'utilisation informatique de la faculté, c'est dans cette optique que le CIPL encadre des recherches en informatique appliquées, gère les salles informatiques et développe des applications pour les services de la faculté. Le service s'occupe aussi de publier les travaux du LASLA et de la R.I.S.S.H. (Revue Informatique et Statistiques dans les Sciences Humaines). C'est dans cette unité que j'ai pu réaliser mon stage, tuteuré par Laurent SIMON, développeur au sein du centre.

2.3. PRESENTATION DE LA MISSION

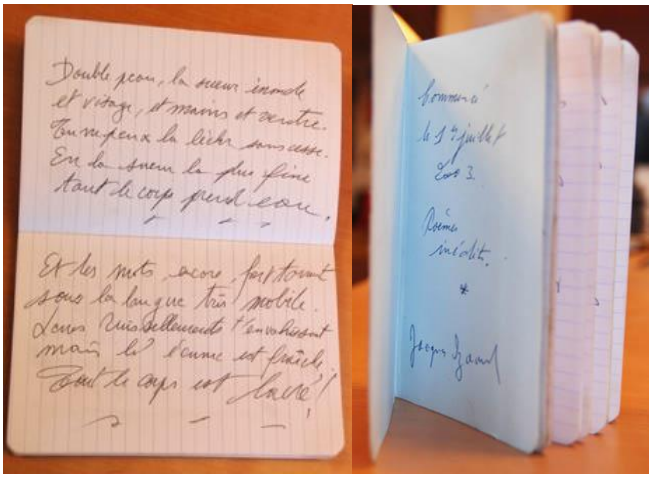
2.3.1. Présentation du travail de M. Purnelle

Il me semblait difficile de présenter ma mission, sans pouvoir l'inscrire dans le travail de M. Purnelle, en effet c'est en vue de l'aider dans ses recherches que le logiciel a été réalisé. M. Purnelle est docteur en philosophie et lettres, directeur du CIPL et directeur adjoint du LASLA. Dans le cadre de ses travaux ce philologue a notamment travaillé sur des thèmes liés à l'analyse lexicographique d'œuvres latines et grecques, la statistique textuelle, la poésie dans sa dimension historique, et linguistique. Domaines et disciplines qu'il enseigne par ailleurs.

En dehors de ses activités universitaires, il est également éditeur aux éditions Le Taillis Pré, dans lesquels il codirige la collection « Ha ! » avec Karel Logist et Yves Namur tout deux poètes. Cette collection regroupe des œuvres de poètes belges contemporains méconnus ou oubliés du public avertis. C'est dans cette optique que M. Purnelle continue à éplucher les archives des poètes Izoard et François Jacqmin :

« *Autant de sources documentaires qui ont servi à établir le texte des œuvres poétiques et permettront d'étudier sa vie, ses relations avec les autres poètes, éditeurs et amis. D'étudier, aussi, la genèse de ses textes, ou simplement de les dater, les identifier, retracer l'histoire des recueils. Le fonds recèle encore bien des éléments susceptibles de nourrir l'étude de l'oeuvre : génétique des poèmes, études des variantes, des modes d'écritures, etc*»
[citation tirée de l'article «Les archives de Jacques Izoard» du site http://culture.ulg.ac.be/]

Exemple de carnet que l'on peut retrouver dans les archives Izoard.



2.3.2. Présentation de François Jacqmin et d'Izoard

- François Jacqmin né en 1929 et mort en 1992, est un poète belge ayant vecu son enfance en Angleterre à la suite du debut de la seconde guerre mondiale. Il y découvrira la psychanalyse Freudienne et il y fit ses premières armes en littérature en langue anglaise. Une fois revenu il rencontrera lors de son service militaire Jospeh Noiret activiste et pratiquant du suréalisme qui créa avec Christian Dotremont notament le mouvement CoBRA ainsi que la revue Phantomas auquel se joindra Jacqmin. C'est dans cette revue que paraitra ses premiers écrits. Ses poèmes y ont une tournure philosophique, classique, et psychologique. Timide et discret, il évitera durant son parcours le carrièrisme littéraire, et ne publiera qu'à l'âge de 50ans son premier recueil. Adepte du mot juste Jacqmin publia peu, travaillant ces poèmes comme le font les orfèvres.

La totalité de ses écrits et brouillons sont actuellement conservés aux Archives et Musée de la Littérature à Bruxelles.

En quelques recueils : « Les Saisons », « Le livre de la neige ».

- Jacques Delmotte est un poète Liegeois né en 1936 et décédé en 2008, il choisit le pseudonyme d'Izoard suite à une promenade dans le col français portant le même nom. Très attaché à sa ville natale, il fît couler beaucoup d'encre à son propos, décrivant la ville, ses jardins, ses escaliers comme nul autre ne l'eut fait. Passioné de poésie, chaque jour était une occasion pour lui d'écrire de ses mots charnels, surréalistes et singuliers aux sonorités sifflantes et travaillées . Il laissat dérrière lui une œuvre abondante couvrant une cinquante de recueils publiés en Belgique comme à l'étranger. Animateur actif et mentor de « l'école de Liège », il permit à de nombreux poètes de se faire connaitre notamment par le biais de rencontres organisées mais aussi via la création de la revue Odradek.

Acquis par la ville de Liège les archives Izoard sont actuellement entreprosées à La Bibliothèque Ulysse Capitaine.

En quelques recueils :

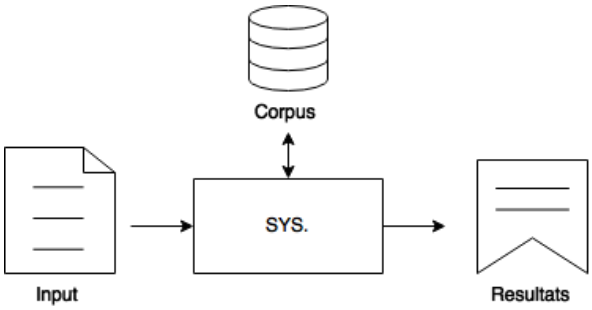
« Voix, vêtements, saccages », « La Patrie empaillée », « Vêtu, dévêtu, libre», « Corps, maisons, tumultes », « Le Bleu et la Poussière » ,« Dormir sept ans ».

3. LA GESTION DU PROJET

3.1. CAHIER DES CHARGES

L'outil demandé devait permettre à M. Purnelle de repérer les textes inédits des archives des deux poètes présentés ci-dessus. A la suite de la numérisation des textes déjà publiés, ainsi compilés sous forme de corpus (travail effectué par le philologue pour les poèmes d'Izoard), le logiciel devait permettre de repérer les textes se rapprochant le plus d'une requête afin de savoir si le manuscrit d'entré valait donc le coup d'être par la suite présent dans un nouveau recueil d'inédits à éditer. Les textes considérés comme inédits sont ceux dont le vocabulaire diffère des poèmes présents dans le corpus, les manuscrits pouvant être aussi des brouillons de poèmes déjà édités ou des versions d'un poème déjà publié.

Le système devait permettre d'entrer plusieurs types de textes en input (mots significatifs, poèmes entiers ou petit corpus sous forme de fichier texte brut, segment de poèmes), de paramétrer des contraintes permettant de limiter le nombre de résultats, et d'afficher après calcul la liste des résultats. Cette liste devait être exportable, et devait implémenter un certain nombre d'indicateurs définis par le linguiste. Enfin, l'interface devait permettre de visualiser via un simple coup d'œil ces fameuses ressemblances via une mise en forme des textes.



Il a donc fallut se mettre d'accord avec M. Purnelle sur le choix des indicateurs à implémenter. Ceux-ci devant permettre une large utilisation du logiciel, afin qu'ils soient transposables aux poèmes de Jacqmin, dont la méthode d'écriture est très différente de celle d'Izoard. En effet, il s'avère qu'Izoard écrivait des poèmes assez stables alors que Jacqmin remaniait plusieurs fois les siens avant qu'ils n'acquièrent leur forme définitive.

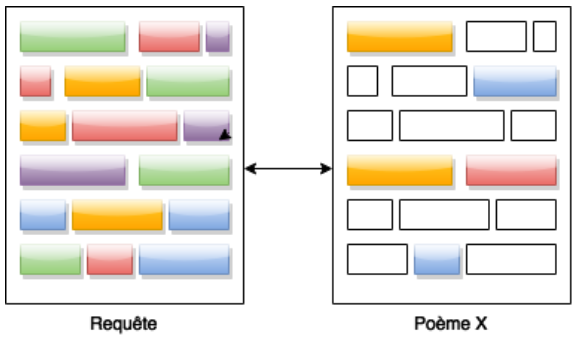


Illustration d'un poème ayant des mots en commun avec la requête, les rectangles imagent les différents mots des deux textes, une couleur représente un mot spécifique

Les indicateurs demandés ont été les suivants :

- Le nombre de termes de la requête présents dans un poème : 5 dans l'exemple
- Le nombre de termes de la requête distincts présents dans un poème : 3 dans l'exemple (le terme jaune, le bleu et le rouge).
- La distance moyenne de ces termes dans le poème : dans notre exemple la distance moyenne est de 3.75. Vous noterez que si la requête exacte se trouve dans un poème, la distance moyenne sera égale à 1.
- Un indicateur de similarité permettant de mesurer à quel point le poème est semblable à la requête, que nous nommerons le taux de similarité. Il est calculé en faisant le rapport du nombre de termes de la requête présents dans un poème avec le nombre de mots total de celui-ci : dans notre exemple le taux de similarité est de 29.4%
  (5 occurrences / 17 mots au total).
- Un indicateur permettant de mesurer l'efficacité de la requête sur un poème donné, que nous nommerons le taux d'efficacité. Il est calculé en faisant le rapport du nombre de termes distincts de la requête présents dans un poème avec nombre distincts de mots présents dans la requête de celui-ci : dans notre exemple le taux de similarité est de 60% (3 occurrences distinctes dans le poème / 5 mots distincts dans la requête).

3.2. ANALYSE DE L'EXISTANT

Le programme demandé était en réalité une amélioration d'un programme réalisé auparavant par Monsieur Simon mon tuteur. Celui-ci était installé sur le serveur de la faculté et était implémenté en php, il permettait d'entrer un certain nombre de recueils afin de les comparer avec une requête textuelle entrée par l'utilisateur. Il retournait par la suite la liste des poèmes dont les termes de la requête étaient mis en valeur par ordre de pertinence (classés selon le nombre de termes de la requête présents dans chaque poème). Celui-ci faisait appel notamment à un anti-dictionnaire permettant de ne pas prendre en compte les mots très fréquents. C'est en m'appuyant sur cette base que j'ai pu entrevoir comment traiter les données textuelles d'entrée et de sortie et ainsi la manière de mettre en place mes requêtes.

3.3. CHOIX DES TECHNOLOGIES

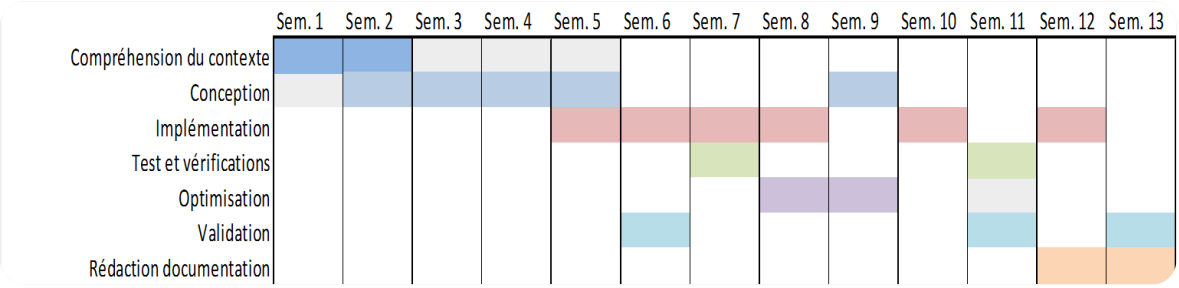
Afin de concevoir le programme il fallait avant tout se positionner sur les technologies qui allaient être utilisées pour l'implémenter. Nous avons donc dans un premier temps, discuté du langage de programmation dans lequel celui-ci allait être codé.

Il fallait que celui-ci puisse être maintenu et donc maitrisé par mon tuteur et moi-même, nous nous sommes donc rapidement penchés sur les langages tels que Ruby et Java, car tous deux sont portables et possèdent des destructeurs natifs.

Ayant travaillé en Java durant mes années de formation, nous nous sommes accordés sur celui-ci, ce choix me permit de me mettre rapidement au travail et de ne pas perdre trop de temps quant à la remise à niveau. Afin de développer le programme, j'ai choisi d'utiliser NetBeans, qui offre une certaine aisance lors de la création de l'interface, ainsi qu'un plugin de monitoring intuitif, permettant de mesurer facilement les performances d'une application en vue de son optimisation. Quant à la base données, je me suis tourné vers le SGBD derby d'Apache, une base de données légère et embarquable.

3.4. DEROULEMENT DE LA MISSION

Pour réaliser ce projet j'ai dû m'imposer une certaine discipline. Je me suis donc dans un premier temps créé un planning global que j'ai dû réajuster chaque semaine, à cause de certaines tâches plus épineuses que prévues. C'est en fonction de ce planning hebdomadaire, que je me fixais chaque jour une liste d'objectifs à atteindre. Le diagramme de Gantt ci-dessous est donc l'image de ce qui a été mis en place à la suite de tous ces ajustements.



La suite de ce rapport est consacrée au détail de la conception, de l'implémentation, des tests effectués ainsi qu'à l'optimisation du code. La compréhension du contexte ayant été présentée plus tôt dans ce rapport.

4. MISE EN OEUVRE DU PROGRAMME

4.1. CONCEPTION

4.1.1. Stratégie globale de la démarche

Notre démarche s'inscrit dans le domaine de la fouille de textes, plus précisément dans une démarche de recherche et d'extraction d'informations. En effet, comme illustré dans le cahier des charges nous devons répondre à une requête non structurée afin de soumettre à notre utilisateur une liste de documents (poèmes) classés par ordre de vraisemblance ainsi que mettre en place une interface de création de corpus à base de fichiers formatés.

Afin d'analyser nos requêtes textuelles, il fallut dans un premier temps choisir l'unité linguistique sur laquelle s'appuyer. Pour cela, nous avons fait le choix de passer par un découpage en sacs de mots, appelé aussi tokens, que j'ai par abus de langage nommé n-gramme dans l'interface du programme. Ces sacs de mots sont des séquences de caractères compris entre deux séparateurs (espaces, tirets…) et se basent donc sur une unité « mot ». Nous avons fait le choix d'extraire tous les sacs de mots composés de 1, 2 et 3 mots, parmi la liste de mots de ce poème, classés par ordre d'apparition. De cette manière, il peut être possible pour le linguiste d'utiliser le logiciel pour repérer des expressions récurrentes.

Le comptage du vocabulaire du corpus découpé en unités linguistiques, constitue le principal critère de pertinence d'un poème quand il est comparé à son nombre de mots total. Cependant cet indicateur semblait trop empirique, car il fallait prendre en compte la structure de celui-ci. Nous avons donc fait le choix de proposer d'implémenter les scores tf-idf et okapi.

4.1.2. Choix de la structure des fichiers poèmes et recueils

Trouver le format du fichier d'input fût la première étape du projet, nous avons dût trouver une solution viable avec M. Purnelle afin d'automatiser le traitement des poèmes par le programme. C'est ainsi que nous en sommes venus à nous accorder sur une mise gestion par fichiers textes (format txt) structurés par des balises et encodés en UTF-8. Afin de mettre en œuvre rapidement cette mise en forme sur le corpus fournit, j'ai réalisé un petit script sous R qui m'a permis d'appliquer cette mise en forme rapidement sur le corpus d'Izoard.

Structure d'un poème et d'un recueil en annexe A et script R en annexe B

4.1.3. Conception de la base de données

Afin de pouvoir gérer notre corpus en local, j'ai fait le choix d'utiliser (comme exprimé dans la partie 3.3. une base de données derby). Celle-ci contient donc 4 tables :

- La table Auteur, contenant le nom, prénom et pseudonyme de chaque auteur de la base de données, sachant qu'un auteur peut être identifié par son nom et prénom ou par un pseudonyme.
- La table Recueil, contenant le titre et l'abréviation de chaque recueil. Le philologue choisi l'abréviation de chacun de ces recueils lors de la création de ceux-ci sous forme de fichier texte. Dans le cadre de notre programme, un recueil n'est écrit par un seul Auteur, et contient au moins un poème.
- La table Poème : où chaque entrée poème est identifiée par un titre, un numéro de recueil, correspondant à son rang d'apparition dans un recueil. Chaque poème possède un contenu, celui-ci étant formaté de la même manière que le fichier texte dont il hérite, ainsi qu'un incipit, correspondant aux 100 premiers mots de celui-ci. Nous noterons qu'un poème peut apparaitre dans plusieurs recueils sous plusieurs formes différentes, l'un pouvant être une nouvelle version d'un autre.
- La table SacDeMots : dans laquelle sont enregistrés tous les patterns composés de 1, 2 et 3mots de chaque poème en minuscule. Cette table nous permet de gagner du temps lors du calcul des résultats d'une requête, elle contient une variable n, correspondant au nombre de mots contenus dans le n-gramme, ainsi que sa position. Pour chaque poème importé le logiciel extrait et formate tous les 1-grammes (sac de mot composé d'un seul mot), 2-grammes (sac de mots composé de deux mots) et 3-grammes (sac de mots composé de trois mots) contenu dans celui-ci.

Afin d'améliorer la recherche d'un tuple dans ces tables, des index sur le contenu de celles-ci ont été créés.

Modèle entité-relation en Annexe C et Script SQL en Annexe D

4.1.4. Le modèle Entité-Objet

Afin de manipuler les entités de la base de données comme des objets j'ai utilisé la librairie JPA. Cette librairie génère pour chaque table de la base de données une classe permettant de gérer les entités de la base. La couche JPA gère la connexion du programme, facilite l'appel des fonctions de gestion de la base données, et permet d'interroger notre base via l'utilisation du langage JPQL, très semblable au SQL.

Architecture de JPA et gestion des objets de la base via JPA en Annexe F & G

Les classes Auteurs, Recueil, Poème et SacDeMots sont ainsi les moules permettant de formater nos données avant leur entrée en base. Elles contiennent les différentes variables liées aux champs de la table de la base leurs correspondant, les getters et setters permettant l'accès et la modification de leurs données. Elles regroupent toutes les méthodes permettant la création de ces objets à partir d'un texte formaté, les méthodes d'insertion, modification et suppression en base. Ces classes sont annotées de manière spécifique guidant JPA dans l'analyse du rôle de chacune des instances de celles-ci. Elles contiennent des requêtes nommées visibles de l'extérieur permettant d'effectuer aisément les actions usuelles comme la recherche d'un objet via son id.

Diagramme de classe des classes en annexe E

4.1.5. Stratégies de réduction de la liste de résultats

Dans l'optique de faciliter le travail de l'utilisateur lors l'analyse des résultats obtenus, à la suite d'une requête, il fallut mettre en place plusieurs solutions :

- Un anti-dictionnaire, modélisé sous forme d'une classe. Il permet d'entrer les mots qui ne seront pas pris en compte dans la recherche. En effet, certains mots sont très récurrents et ne donnent pas de sens à la recherche. Modélisé de manière à pouvoir générer automatiquement des listes de « mots-vides », cette classe permet de récupérer les mots les plus fréquents dans l'œuvre d'un auteur, ou dans une liste de recueils.

- Des mesures de similarité, afin d'obtenir un classement performant des résultats. Deux mesures ont été implémentées : le Tf-Idf et le BM25. Ces scores permettent de mesurer la pertinence d'un document en fonction de son contenu face à une requête. Le logiciel n'affichant que les 50 premiers poèmes les plus pertinents, classer les résultats via l'utilisation de l'un de ces indicateurs permet d'effectuer un premier écrémage assez précis.

    - Le score TF-IDF : est un calcul de poids prenant en compte le nombre d'occurrences trouvées dans un poème (TF) ainsi que le poids de ce terme dans le corpus (IDF), dans la mesure où plus un terme apparait dans un corpus moins il parait pertinent.
    - Okapi BM25 : Cet indicateur est une version pondérée du tf-idf prenant en compte la longueur des documents, car nous avons dans un document long plus de chance de trouver un terme souhaité.

- Les paramètres de contraintes, englobés dans la classe Parametres_requete, offrent la possibilité à l'utilisateur de contraindre l'affichage des résultats selon des attentes numériques :
    - Celles basées sur un minimum à atteindre

- le nombre d'occurrences minimum
- nombre d'occurrences distinctes minimum
- le nombre de distances égale à 1.
- Le taux d'efficacité (à fournir en pourcentage)
- Le taux de ressemblance (à fournir en pourcentage)

    - Celles basées sur un maximum à atteindre

- la distance maximale entre deux occurrences
- la distance moyenne entre toutes les occurrences du poème
- la distance minimale entre deux occurrences

Le détail du calcul des indicateurs implémenté est en Annexe H

4.1.6. Classes liées aux traitements des requêtes

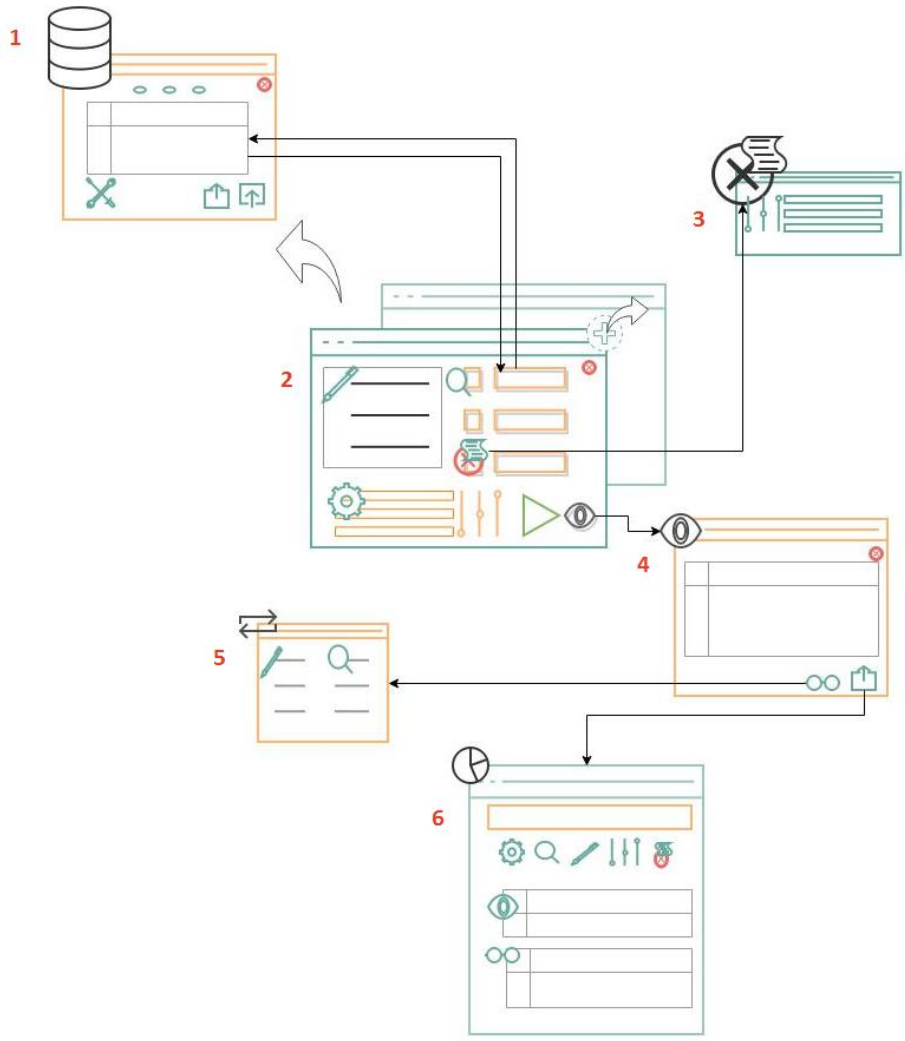
Une fois que les paramètres des requêtes ont été définis, j'ai conçu quatre classes, héritant d'une classe abstraite nommée Requete. Celles-ci permettant de gérer les différents types de textes entrés par l'utilisateur : les requêtes basés sur un fichier formaté (contenant un recueil ou poème), ainsi que les requêtes basés sur des entrées utilisateurs (segment de texte, ou par mots significatifs).

Ces classes enregistrent les paramètres, le champ de recherche de la requête (l'auteur et les recueils concernés par la recherche, préalablement définis par l'utilisateur) et permettent de lancer le calcul des résultats afin de les extraire par la suite.

Le diagramme de classe associé aux requêtes est en Annexe I

4.1.7. Modélisation de l'interface

Après avoir modélisé les cas d'utilisation et les premières briques de notre programme il fallait concevoir l'interface du programme. Afin d'avoir une vue globale des parcours possible par l'utilisateur.



Le programme se modélise dans un premier temps sous la forme d'une fenêtre principale (2), permettant de créer des requêtes, de les paramétrer. Une fois que la requête lancée est calculée l'utilisateur visualise les résultats sous forme tabulaire(4), cette fenêtre lui permet par la suite de visualiser le poème en question (5) ou d'extraire la liste des résultats sous la forme d'un rapport (6). L'utilisateur a aussi la possibilité de gérer le corpus de base de données via une interface prévue à l'effet (1).

Diagramme de cas d'utilisations en Annexe J, K & L

4.2. IMPLEMENTATION

Après avoir présenté la modélisation de notre programme au niveau de son architecture, de son design et de ses fonctionnalités, place à son implémentation. Comme nous l'avons vu le programme possède trois fenêtres principales auxquelles sont liées les différentes fonctionnalités de celui-ci : une fenêtre de création de requêtes, une fenêtre de visualisation de résultats ainsi qu'une fenêtre de gestion de base de données. Pour chacune de ces fenêtres il a donc fallu gérer tous les évènements liés aux actions de l'utilisateur et placer les algorithmes de calcul en fond de taches dans des classes héritant de la classe SwingWorker. Nous y avons placé les algorithmes destinés à la récupération des données de la base, les algorithmes de mise en forme et de manipulation des données.

Nous exposons dans cette partie les principaux de notre programme, notamment ceux liés à la gestion de la base de données, à la création automatique d'anti-dictionnaire, aux calculs de requêtes, à la mise en forme des poèmes ainsi qu'à la création de rapports.

4.2.1. Gestion de la base de données

Afin de permettre à l'utilisateur de gérer sa base de données de nombreuses solutions ont été mise en place, offrant à l'utilisateur une utilisation flexible du logiciel. La gestion des données se fait via l'utilisation d'une interface dédiée, laissant le choix à l'utilisateur de gérer son corpus de deux manières : une gestion via des fichiers textes (import de fichiers et export), ainsi qu'une gestion manuelle (création d'entités, via le remplissage de boites de dialogue).

Revenons à nos quatre classes « moules » (Annexe E), que sont les classes Auteur, Recueil, Poème et SacDeMots, elles sont dans ce cas d'utilisation les principales actrices. En effet, étant le pont entre la base et notre programme, elles contiennent toutes les fonctions liées à la transformation d'un fichier au format .txt en entité, ainsi que celles liées à l'import, export, la modification et la suppression. Vous noterez qu'elles ont été configurées de manière à gérer les entités en cascade, propageant ainsi toute modification d'une classe mère à sa classe fille. La classe SacDeMots, n'est pas visible à l'utilisateur, toutes les actions émises sur les objets de cette classe découlent donc d'une action sur un objet de la classe mère Poème.

Comme nous l'avons dit précédemment, ces classes réalisent la transformation d'un fichier texte en entité de la base de données. En effet, les classes recueils et poèmes se reposent sur la structuration des fichiers d'entrée pour extraire les données liées à celle-ci.

La fonction SetFromTxt() de la classe poème, extrait le titre d'un poème, son numéro, son incipit ainsi le contenu du poème entré et formaté sous la forme d'une chaine de caractères. Il découpe par la suite le contenu de celui-ci en 1-grammes (sac de mots composé d'un seul mot), 2-grammes et 3-grammes formant de nouvelles entités de la classe SacDeMots.

La fonction SetFromTxt() de la classe recueil, repère dans la même optique le titre d'un recueil d'entré ,sous forme de chaîne de caractères, ainsi que l'emplacement de chacune des balises indiquant le titre de chacun de ses poèmes, extrayant de cette manière la liste de poèmes du recueil par le biais de la fonction SetFromTxt() de la classe poème.

Ont été aussi implémentés des fonctions de suppression d'entités (RemoveInBdd()), de modification d'entités (UpdateInBdd()), et d'insertion InsertionBdd().

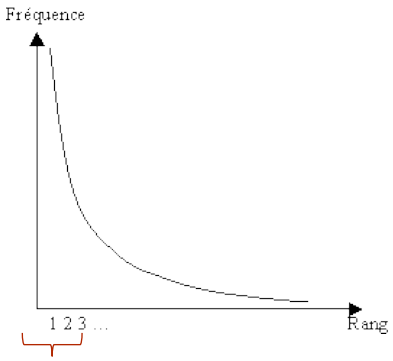
Algorithmes 1,2,3 en Annexe M,N & O

4.2.2. Anti-dictionnaire

La classe anti-dictionnaire possède une fonction de création d'anti-dictionnaire à partir d'un fichier, récupérant de cette manière la liste des mots « vides », ainsi qu'une fonction permettant l'export au format texte de celui-ci. Cet export comprend alors la liste des mots le composant séparés d'une virgule.

Nous avons évoqué précédemment le mode « automatique », celui-ci est géré par les procédures ImplementFromAuteurRecueil() et ImplementFromAuteur(). Ces fonctions ont vocation à proposer un anti-dictionnaire à l'utilisateur, sur la base d'un auteur ou d'une liste de recueils. Elles recherchent alors dans les œuvres fournies les segments les plus fréquents. Concrètement ces procédures trient les figures (1-grammes, 2-grammes ou 3-grammes selon le paramétrage de la requête en cours) des ouvrages selon leur fréquence documentaire dans une liste, en choisissant un seuil compris entre 0 et 5% ; l'utilisateur obtient la part correspondant au seuil sélectionné de la liste des « mots vides » classée par fréquence d'apparition.

C'est sur le concept de la loi de Zipf que s'appuie donc cette fonctionnalité. En effet certains mots trop fréquents ne portent aucune valeur sémantique. Ce sont traditionnellement le cas des mots grammaticaux (déterminants, prépositions, conjonctions…), des auxiliaires, des verbes supports (comme « faire », « prendre »).



Seuil définit

Cette fonctionnalité détournée de son but original, permet au philologue de trouver les mots fréquemment utilisés dans un corpus qui ne sont pas des mots outils afin de les utiliser par la suite dans une requête.

Algorithme 4en Annexe P

4.2.3. Le calcul des résultats

Une fois que l'utilisateur a fini de paramétrer sa requête, il lance la procédure de calcul via un bouton. Le logiciel récupère alors tous les paramètres renseignés par l'utilisateur et formate la requête textuelle selon son type. Il créé alors une instance d'une classe héritant de Requete du type de requête sélectionnée et en appelle à la procédure LunchCalcResultats() de cet objet via le thread nommé CALCUL_Worker.

Cette procédure récupère via une requête JPQL tous les contenus et positions des sacs de mots concernés par les paramètres entrés dans une grande table implémentant pour chaque poème les indicateurs de la classe Result. Elle passe en revue tous les sacs de mots de cette table et compare ceux-ci avec des termes présents dans la requête, le programme agrège pour chaque poème le nombre d'occurrences de ces termes, le nombre d'occurrences distinctes, calcule la distance moyenne de ces termes et enregistre ces indicateurs s'ils respectent les contraintes définies par l'utilisateur. Il est important de mentionner le fait que dans le cas d'une requête basée sur des mots significatifs (instance de Requete_Mots), les indicateurs liés à la distance entre deux occurrences d'un terme de la requête dans un poème ne sont pas calculés.

A la suite de cela le programme parcours la liste des résultats, des objets de la classes Result, retenus et calcule pour chacun des poèmes identifiés par ces objets les scores Okapi et Tf-Idf de ceux-ci, afin de les classer par ordre de pertinence cette liste qui sera par la suite tronquée à 50 résultats.

Vous noterez que les requêtes de types Requete_Recueils réalisent cette procédure pour chacun des poèmes du recueil d'entrée.

4.2.4. Mise en forme des poèmes

Après avoir terminé le calcul d'une requête le programme offre une vue tabulaire de tous les résultats par poème. Le linguiste peux alors en sélectionnant un poème qui lui semble intéressant le comparer avec la requête d'entrée. Une fenêtre apparait alors et mets face à face les deux textes en surlignant les occurrences communes à ceux-ci. Un encart en bas de cette fenêtre liste ces occurrences ainsi que leur fréquence d'apparition dans le poème sélectionné.

Deux versions de la procédure de mise en forme ont été implémentées dans la classe MiseEnForme :

La première, parcours la liste des mots communs et recherche si il existe une occurrence de ce mots dans la chaine fournies par la requête puis réalise la même procédure pour le texte du poème sélectionné. Si une occurrence de ce motif apparait dans le texte, le logiciel vérifie alors si celui-ci n'est pas un segment d'un autre mot dans le texte (exemple le motif « la » se retrouvant dans le mot « lapin »), et si ce n'est pas le cas le formate en ajoutant à sa droite et sa gauche les balises Html de mise en forme. C'est en implémentant les règles de typographie que cela a été possible. Performant appliqué à des requête se basant sur des 1-grammes (sac de mots composé d'un seul mot), la mise en forme est délicate dans le cas des n-grammes, surtout dans le cas où ceux-ci se chevauchent dans le poème.

La seconde a donc été implémentée afin de rendre possible le formatage pour les 2-grammes et 3-grammes, en effet lors de leurs entrées en base ces sacs de mots sont vidés de tout symboles et mise en minuscules. Il alors très difficile de s'appuyer sur les règles typographiques pour repérer ces figures en gardant la mise en forme originale du texte. Concrètement l'algorithme parcours la liste de sac de mots de la requête et enregistre chaque mot dans une chaine soit par sa forme originale soit par celui-ci mise en forme si il appartient à la liste des mots communs. Cette opération est répétée par la suite sur la liste de sac de mots formatés du poème courant. Le formatage résultant de cette procédure est moins élégant que par la première méthode, car elle formate un texte en minuscule sans ponctuations… Cependant son implémentation permet de fournir une solution non optimale mais suffisante, à un cas délicat de traitement.

Algorithme 5 en Annexe Q

4.2.5. Export et création de rapports

Dans l'optique de créer un rapport lisible sur toutes les plateformes, il a été choisi d'utiliser le format HTML. Lors de la génération d'un rapport le logiciel se base sur un fichier template, dont il remplace les champs par les paramètres de la requête réalisé, et présente les résultats obtenus sous la forme de tableau, l'un présentant par poèmes les indicateurs calculé et l'autre le comparatifs entre les poèmes et la requêtes mis en forme via l'utilisation de la classe MiseEnForme présentée ci-dessus (un système d'ancre permet d'acceder à la mise en forme d'un poème depuis sa liste d'indices). En générant ce rapport, l'utilisateur peut réordonner la liste de résultats selon ces préférences et n'extraire qu'une partie de celle-ci.

Template du rapport en Annexe R

4.3. DEBOGAGE, TEST ET OPTIMISATION

Afin de fournir un logiciel performant et fonctionnel il est impossible de ne pas opérer à des phases de débogage et d'optimisation. Ces deux étapes récurrentes dans mon projet vous sont présentées dans cette partie.

4.3.1. Débogage

Lors de l'implémentation d'un projet informatique, il incombe de réaliser très souvent des phases de débogage. Ce fût donc une phase que j'ai opérée dans deux cas :

- Résoudre un bug ponctuel : le bug apparait lorsque l'on compile son code à la suite d'une modification. Dans ce cas de figure c'est donc l'erreur qui viens à nous et il est facile de remonter à la source du problème. Dans ce cas l'utilisation du débuggeur de NetBeans est bien utile puisqu'il nous permet de suivre le cycle des objets instanciés durant le fonctionnement du programme notamment quand un problème se propage.

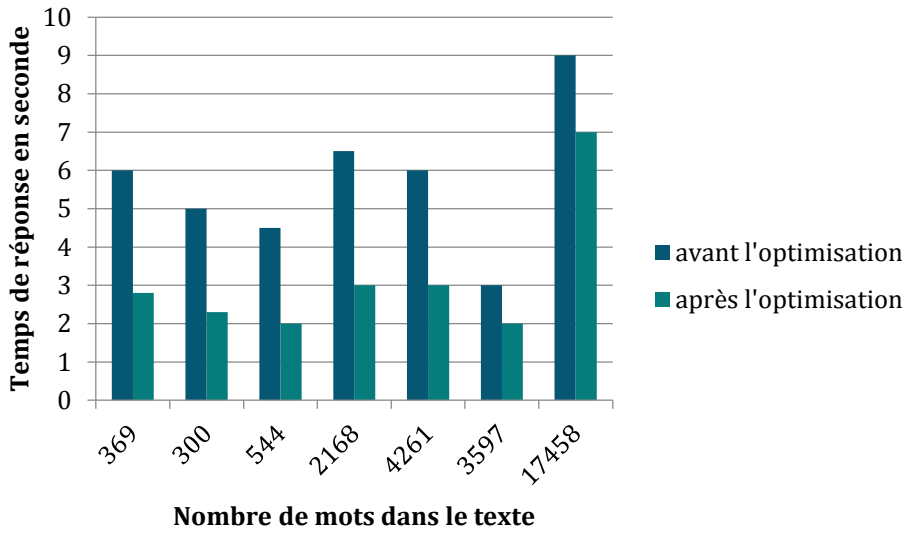
- Le cas de contrôle : le bug est traqué par le développeur. Dans ce cas de figure le concepteur anticipe un cas problématique d'utilisation. Utiliser le framework JUnit me permis d'effectuer les premières vérifications dans ce sens à la fin de la création d'une classe. Il fallut de plus effectuer des tests avec des cas atypiques et anticiper des cas d'utilisation marginaux afin d'assurer la fonctionnalité du programme à long terme.

4.3.2. L'optimisation

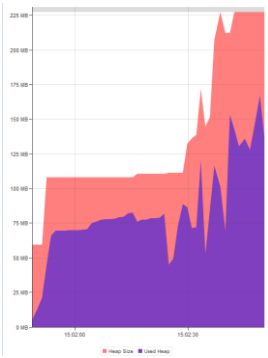
L'optimisation a été essentielle à la réalisation de ce programme notamment en ce qui concerne le traitement des requêtes. En effet il fallait que le programme puisse fournir rapidement les résultats escomptés pour que celui-ci ne perde pas son intérêt.

Afin de se rendre compte des performances de notre programme nous avons chronométré le temps de réponse à une requête, et nous sommes vite rendu compte que le programme passait une grande partie de son temps à effectuer la requête permettant d'extraire les données des sacs de mots concernés par le champ de recherche. Dans le but de remédier à ce problème de performance nous avons donc mis en places plusieurs solutions : la création d'index basés sur le contenu des tûples dans notre base de données, nous avons dû modifier la requête JPQL associée à la création de notre table en la traduisant en SQL natif et pour finir revu l'algorithme entier contenu dans la procédure LunchCalcResultats() des classes héritant de Requête. L'outil de monitoring de Netbeans me fût aussi d'un grand secours, il me permit de visualiser l'évolution de l'état des threads et de la consommation de mémoire au fil du temps.

Ce graphique illustre le gain de performance en temps de l'application après optimisation, de la fonction de réquêtage sur le corpus complet.



Durant cette phase nous avons dut augmenter la capacité de la JVM (Java Virtuel Machine) utilisée durant le programme, afin de permettre le traitement des requêtes basées sur des recueils comportant de nombreux poèmes. En effet la mémoire utilisée durant le lancement d'une de ses requêtes plafonnait vite à 256Mb, comme l'illustre ce graphique tiré d'un test effectué sur un recueil de 50 poèmes sur tout le corpus d'Izoard:



Nous avons donc passé la mémoire de notre machine virtuelle à 512Mb, augmenter ce seuil a permis au logiciel de prendre en entrée plus de poèmes dans un recueil.

Nous retiendrons que le logiciel ne sais pas gérer des requêtes portant sur un recueil trop volumineux, malgré l'augmentation de la mémoire dédiée celui-ci peux difficilement calculer une requête portant sur l'intégralité du corpus si celle-ci est un recueil comportant plus de 80 poèmes (ceci est en grande partie due au fait que les requêtes de type recueils sont des listes de requêtes de type poème). Un message de prévention apparait donc lorsque l'utilisateur lance une requête de ce type. Cependant l'utilisation de corpus aussi volumineux n'arrivera pas dans la pratique, comme me l'a confirmé mon maitre d'œuvre, puisque ces recueils de requetâges seront en réalité générés artificiellement par le philologue afin de tester plusieurs poèmes à la fois.

4.4. VALIDATION

La validation est la phase du projet durant laquelle j'ai pu soumettre mon projet à mon maitre d'œuvre. J'ai donc pu présenter trois fois mon travail :

- Le premier rendez-vous m'a permis de vérifier avec lui les sorties numériques et les résultats fournis par l'algorithme de calcul d'indicateurs, j'ai pu notamment lui présenter le design global de l'appli.
- Le second rendez-vous m'a permis de lui présenter le logiciel quasi-fonctionnel, cette rencontre a fait suite au travail d'optimisation. Ce fût l'occasion pour mon maitre d'œuvre de me fournir une liste de correctifs à réaliser et de me faire un retour sur les performances du programme.
- Le dernier rendez-vous correspond à la démonstration finale de mon programme à mon maître d'œuvre, je lui ai donc montré comment utiliser toutes les fonctionnalités du programme et lui ai présenté les modifications mises en œuvre pour satisfaire sa demande.

4.5. REDACTION DE LA DOCUMENTATION

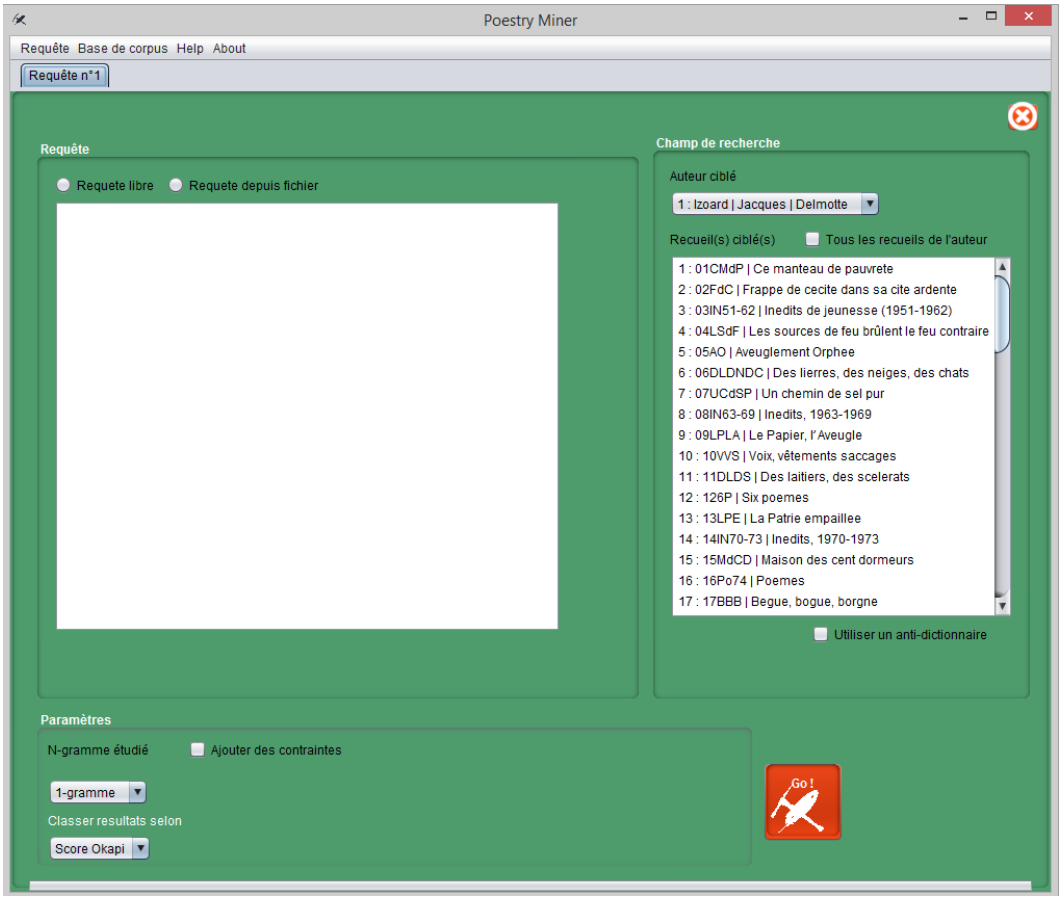
Dans le but de rendre accessible mon travail et permettre prise en main du logiciel à un nouvel utilisateur j'ai réalisé un guide utilisateur intégré au programme. Il présente la procédure de lancement du programme, les différentes fonctionnalités de celui-ci et rappelle notamment les règles de calculs des indicateurs. De plus dans l'optique de maintenir l'outil j'ai commenté tout le code du programme et exporté cela sous forme d'un pdf JavaDoc afin de le fournir à M. Simon mon tuteur. Ces documents participerons je l'espère à la pérennité de mon logiciel dans le temps.

5. LE RESULTAT : POESTRY MINER

Après avoir parcouru l'ensemble des étapes de la réalisation de ma mission, je vous présente dans cette partie le logiciel conçut à l'issue de mon stage nommé Poestry Miner. Nous ferons un rapide tour de ses fonctionnalités, puis nous présenterons ses performances et de ce qui aurait pu être mis en place avec le recul.

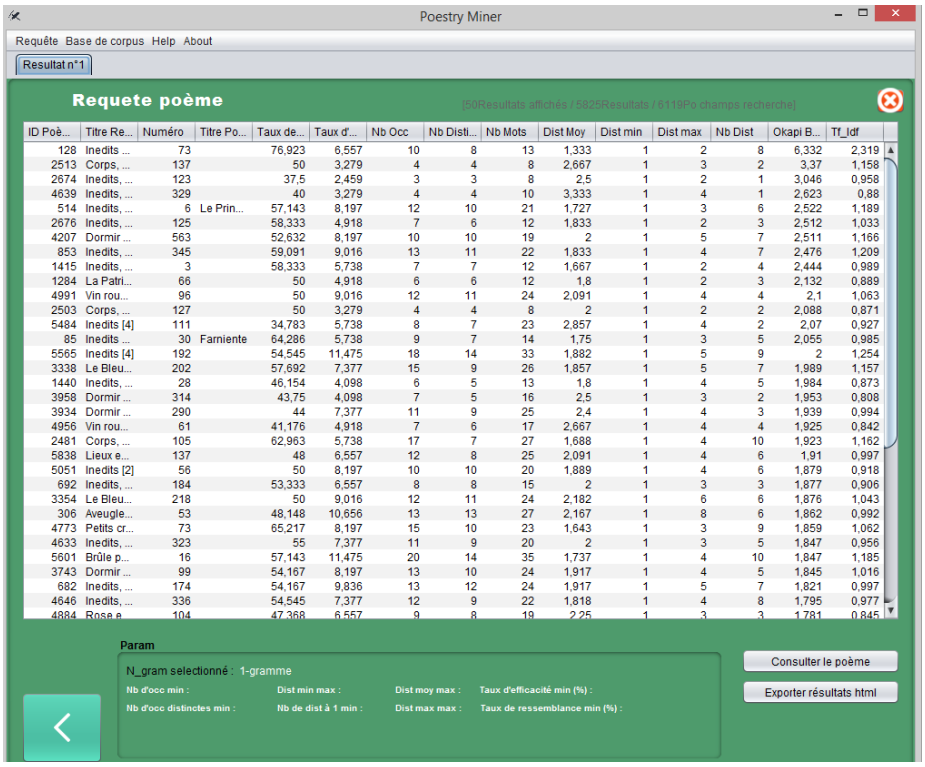
5.1. PRESENTATION DU PROGRAMME

5.1.1. Interface principale

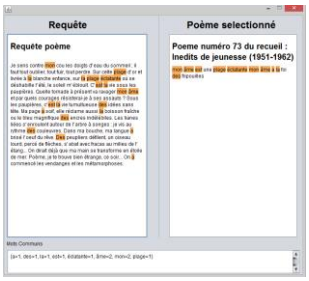


Lors du lancement du programme, une fenêtre comme celle-ci apparait. La barre de menu permet à celui-ci d'ajouter une nouvelle requête, de gérer la base de données, ainsi que d'obtenir de l'aide (guide utilisateur). L'utilisateur depuis le panneau « Requête n°1 » peux configurer une première requête en entrant un texte (à partir d'un fichier ou manuellement), paramétré celle-ci : choisir l'unité d'analyse en sélectionnant un N-gramme, choisir le score de classement à affecter aux résultats, ainsi que contraindre l'affichage des résultats sur un critère numérique. L'encart de gauche lui permet de sélectionner le champ de sa recherche (l'auteur et les recueils) et d'affecter un anti-dictionnaire à la requête via la check-box présente en bas de celui-ci. Un fois le paramétrage terminé, il suffit de lancer la requête via le bouton orange. La progression du calcul est indiquée via une barre de progression, une fois terminé l'utilisateur bascule sur l'interface de résultats.

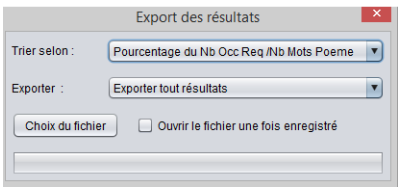
5.1.2. Interface de résultats



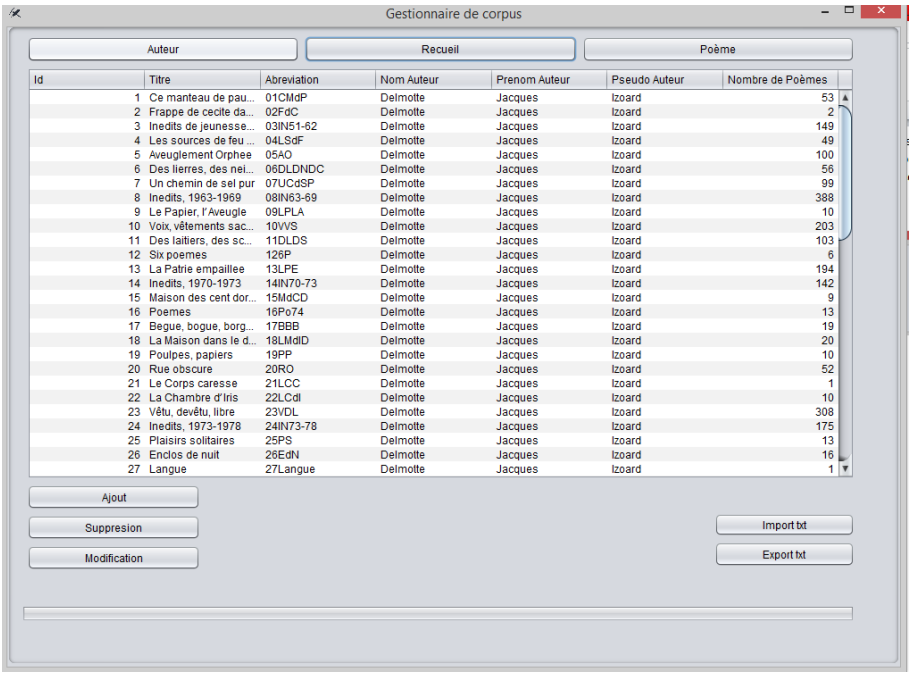
L'interface de résultat rappelle le type de la requête courante et les paramètres lui étant affectés. Elle présente un tableau regroupant les 50 poèmes du corpus les plus vraisemblables, ainsi que de nombreux indicateurs permettant à l'utilisateur de juger de la pertinence de ce résultat. Afin d'analyser en détail un poème, l'utilisateur a la possibilité de cliquer sur l'onglet « Consulter poème » afin d'obtenir un comparatif mis en forme de la requête et de ce poème. Cette fenêtre permet de plus à celui-ci d'obtenir le détail des occurrences ainsi que leur fréquence d'apparition dans le poème sélectionné.



Enfin depuis cette interface il est possible pour l'utilisateur d'enregistrer un rapport HTML rassemblant toutes ces informations en cliquant sur le bouton « Exporter résultats HTML ». Celui-ci peut de plus choisir de l'ouvrir directement dans un navigateur web.



5.1.3. Interface de gestion de corpus



Id	Titre	Abreviation	Nom Auteur	Prenom Auteur	Pseudo Auteur	Nombre de Poèmes
1	Ce manteau de pau...	01CMdP	Delmotte	Jacques	Izoard	53
2	Frappe de cecite da...	02FdC	Delmotte	Jacques	Izoard	2
3	Inedits de jeunesse...	03IN51-62	Delmotte	Jacques	Izoard	149
4	Les sources de feu ...	04LSdF	Delmotte	Jacques	Izoard	49
5	Aveuglement Orphee	05AO	Delmotte	Jacques	Izoard	100
6	Des lierres, des nei...	06DLDNDC	Delmotte	Jacques	Izoard	56
7	Un chemin de sel pur	07UCdSP	Delmotte	Jacques	Izoard	99
8	Inedits, 1963-1969	08IN63-69	Delmotte	Jacques	Izoard	388
9	Le Papier, l'Aveugle	09LPLA	Delmotte	Jacques	Izoard	10
10	Voix, vêtements sac...	10VVS	Delmotte	Jacques	Izoard	203
11	Des laitiers, des sc...	11DLDS	Delmotte	Jacques	Izoard	103
12	Six poemes	126P	Delmotte	Jacques	Izoard	6
13	La Patrie empaillee	13LPE	Delmotte	Jacques	Izoard	194
14	Inedits, 1970-1973	14IN70-73	Delmotte	Jacques	Izoard	142
15	Maison des cent dor...	15MdCD	Delmotte	Jacques	Izoard	9
16	Poemes	16Po74	Delmotte	Jacques	Izoard	13
17	Begue, bogue, borg...	17BBB	Delmotte	Jacques	Izoard	19
18	La Maison dans le d...	18LMdlD	Delmotte	Jacques	Izoard	20
19	Poulpes, papiers	19PP	Delmotte	Jacques	Izoard	10
20	Rue obscure	20RO	Delmotte	Jacques	Izoard	52
21	Le Corps caresse	21LCC	Delmotte	Jacques	Izoard	1
22	La Chambre d'Iris	22LCdI	Delmotte	Jacques	Izoard	10
23	Vêtu, devêtu, libre	23VDL	Delmotte	Jacques	Izoard	308
24	Inedits, 1973-1978	24IN73-78	Delmotte	Jacques	Izoard	175
25	Plaisirs solitaires	25PS	Delmotte	Jacques	Izoard	13
26	Enclos de nuit	26EdN	Delmotte	Jacques	Izoard	16
27	Langue	27Langue	Delmotte	Jacques	Izoard	1

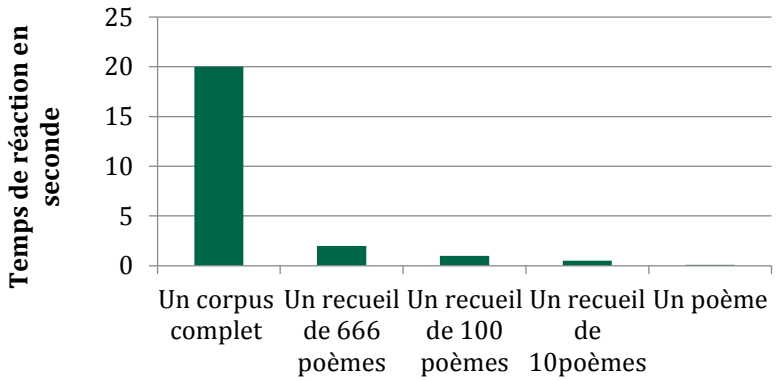
Cette fenêtre peut être appelée depuis la barre de menu, elle offre à l'utilisateur l'opportunité de gérer son corpus, tout d'abord en lui permettant de consulter son contenu. Il a aussi la possibilité d'ajouter, supprimer et modifier manuellement des auteurs, des recueils et des poèmes depuis cette interface. Mais elle lui permet aussi d'importer des fichiers texte directement afin d'entrer un recueil ou une œuvre plus rapidement, tout en lui accordant l'opportunité de les exporter par la suite.

5.2. PERFORMANCES ET LIMITES

Dans le but d'obtenir des mesures pour vous présenter les performances de Poestry Miner, j'ai utilisé l'outil de Monitoring Netbeans qui m'a permis de mesurer le temps de réaction du programme lors de l'export de fichiers, l'import ainsi qu'après le lancement du calcul d'une requête.

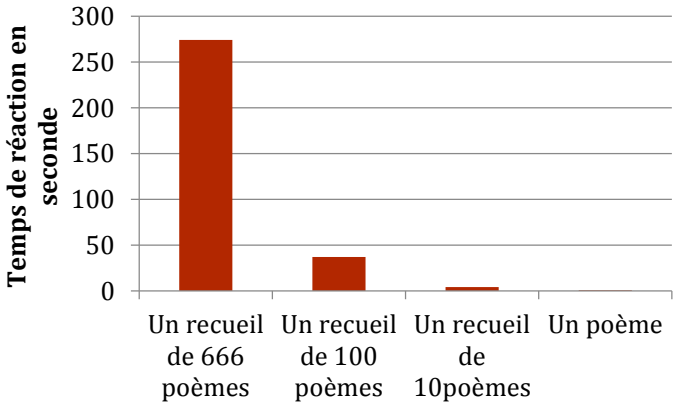
5.2.1. Export de fichier

A l'export de fichier PoestryMiner est très rapide, 1 à 2 secondes pour exporter un recueil et un millième de seconde pour un poème. Extraire le corpus d'Izoard prend par exemple 20 secondes (71 recueils soit 6119 poèmes), comme le montre cet histogramme :



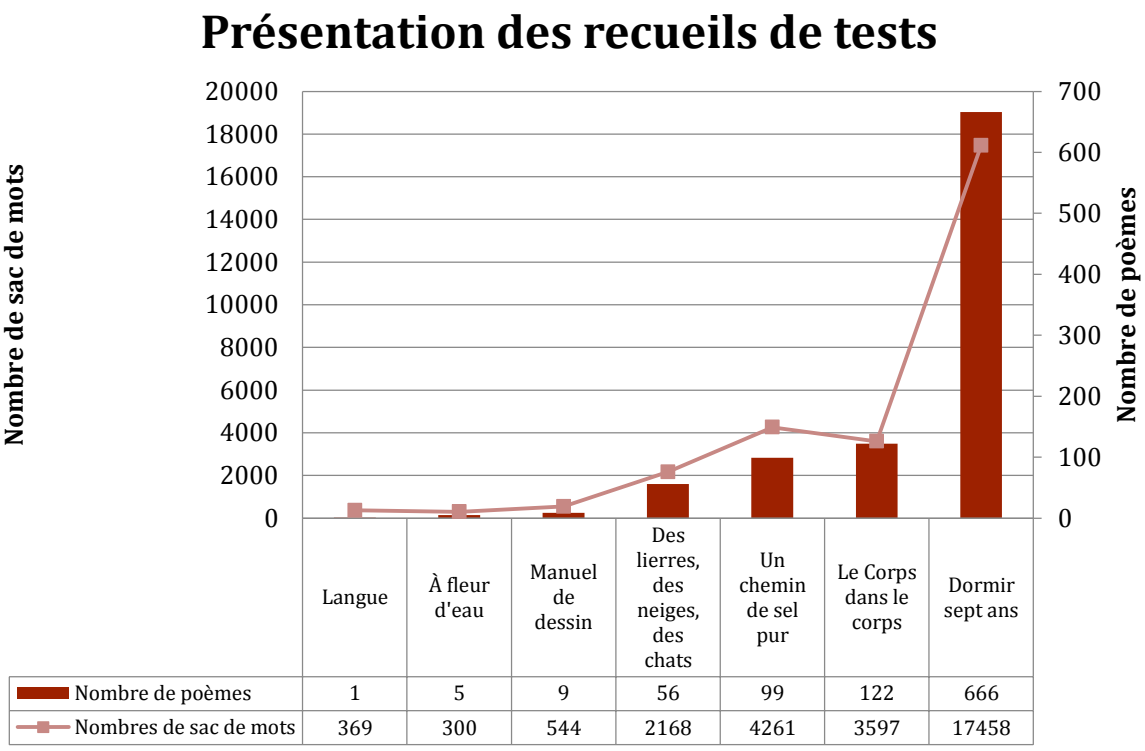
5.2.2. Import de fichier

Importer est une autre affaire, en effet comme le logiciel gère le découpage des poèmes en sac de mots de 1, 2 et 3 mots cela est moins rapide. Il faudra donc compter entre 250 et 4 secondes pour importer un corpus selon le nombre de poèmes qu'il contient, et 0.5 seconde pour un poème seul. Importer l'œuvre complète d'un auteur n'est donc pas très rapide, il faudra compter près d'une heure pour importer les 71 recueils d'Izoard…



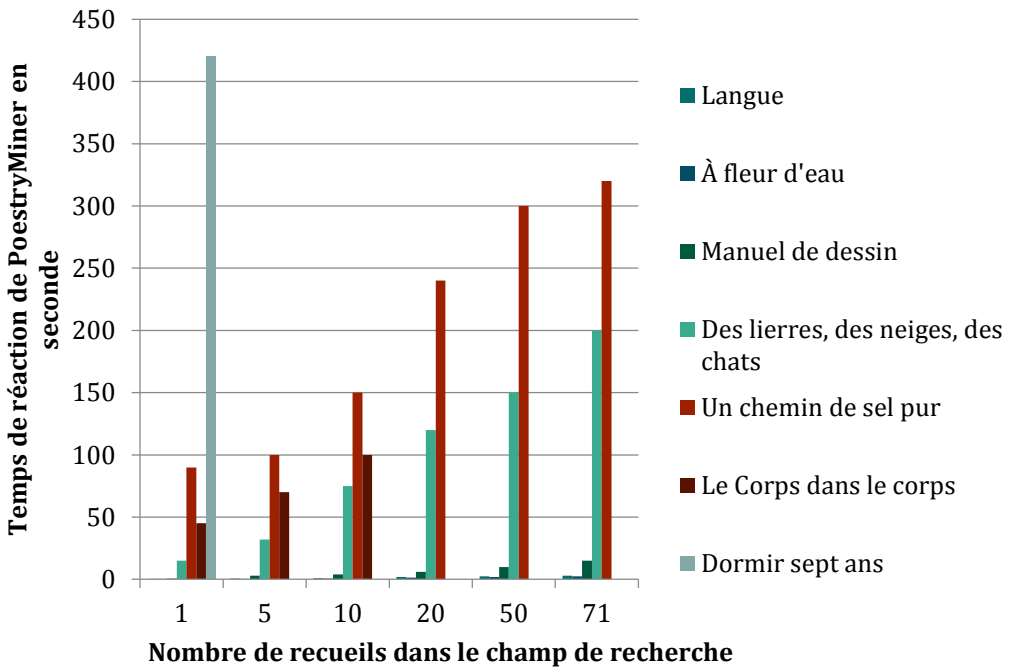
5.2.3. Requêtes de type recueils

Afin de mesurer le temps de réponse de notre algorithme de requête nous avons donc dans un premier temps testé les requêtes de type recueils. Nous avons donc sélectionné 7 recueils parmi le corpus d'Izoard en fonction du nombre de poèmes qu'ils contenaient. En voici une rapide présentation graphique et numérique.



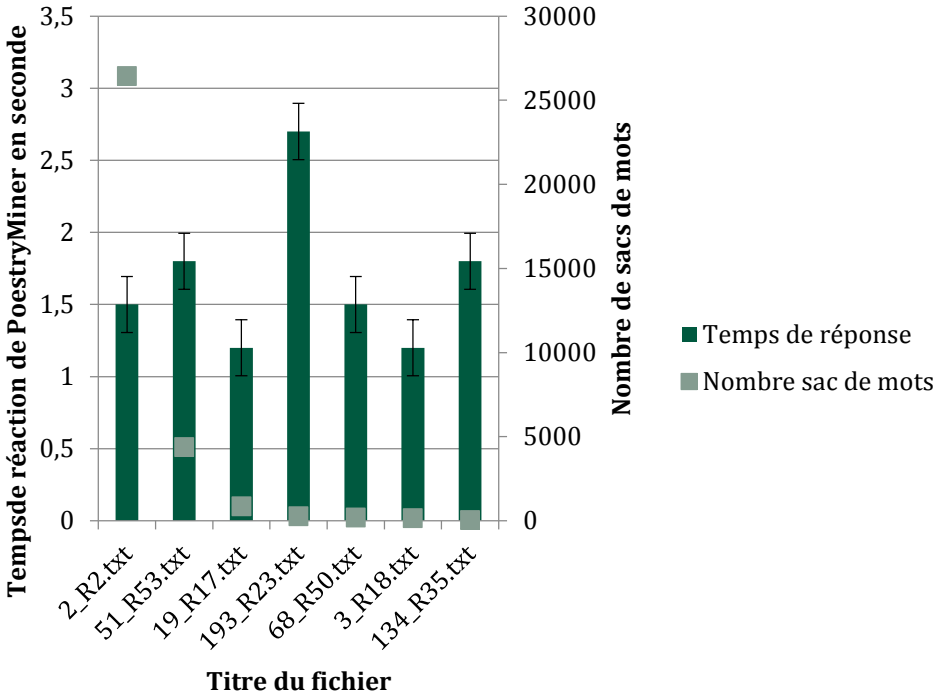
	Langue	À fleur d'eau	Manuel de dessin	Des lierres, des neiges, des chats	Un chemin de sel pur	Le Corps dans le corps	Dormir sept ans
Nombre de poèmes	1	5	9	56	99	122	666
Nombres de sac de mots	369	300	544	2168	4261	3597	17458

Comme nous l'avons exprimé dans la partie dédiée à la présentation de la phase d'optimisation Poestry Miner a du mal à requêter sur de longs recueils. Ce défaut est du a un dépassement mémoire étant notamment atteint pour les recueils de plus de 100 poèmes (ce qui est déjà pas mal). Comme le démontre le graphique suivant les recueils « Le corps dans le corps » et « Dormir sept ans » contenant tous les deux respectivement 122 et 666 poèmes, n'ont pas pu être requêté sur beaucoup de recueils définis dans le champ de recherche. En effet, seulement 10 recueils on pût être affectés en tant que champ de recherche au premier et un seul pour le second. Nous remarquerons cependant les bonnes performances du logiciel sur les petits recueils, fournissant à l'utilisateur un résultat après un maximum de 15 secondes pour un recueil de 10 poèmes.



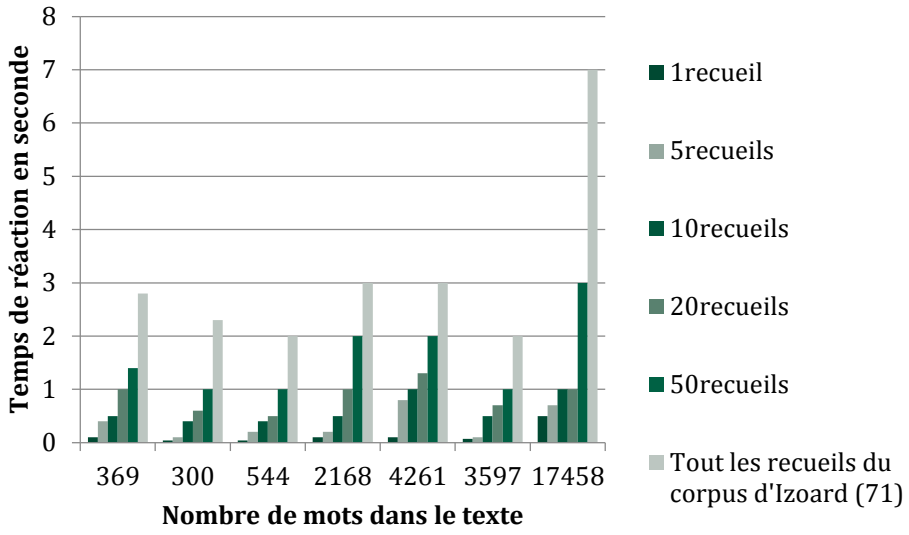
5.2.4. Requêtes de type poèmes

Nous avons ensuite testé les requêtes se basant sur des poèmes, nous avons donc extrait des poèmes plus ou moins longs afin de tester les performances de notre programme. C'est ainsi que nous avons pu mesurer des résultats rassurant. En effet le logiciel répond au maximum après 3 secondes sur le corpus complet d'Izoard.



5.2.5. Requêtes de type texte

Enfin ce graphique présente les performances du programme sur les requêtes sur des requêtes de type texte. Comme vous pouvez le remarquer ces requêtes sont aussi très rapides et prennent entre 2 et 3 secondes avant d'être affichés. Pour de très grand texte (exemple de celui comportant 17458), ceci prendra près de 8seconde (ce qui reste raisonnable).



5.3. POUR ALLER PLUS LOIN

Poestry Miner a su remplir sa mission en fournissant à M. Purnelle une plateforme logicielle lui permettant de gérer des corpus de poèmes en local (via notamment l'utilisation de fichiers textes), de réaliser des requêtes multicritères élaborées, ainsi que d'exporter les résultats s'y rattachant via des fichiers HTML afin de les analyser.

Je suis finalement satisfait des résultats obtenus à l'échéance de ce stage, cependant étant débutant en programmation je me rends compte aujourd'hui que certains choix techniques n'ont pas été optimaux : par exemple il aurait été judicieux de migrer vers une base de données orientée objets comme PostgreSQL qui aurait pût permettre d'accélérer le calcul des requêtes via notre mapping Entité-Objet ou encore d'utiliser la librairie Lucene qui m'aurait permis de mettre en place un indexage plus soigné des poèmes. De plus, j'aurai aimé fournir un code SOLID afin de rendre mon programme plus robuste.

Enfin, je pense qu'il aurait été possible d'étendre l'utilisation de notre programme avec d'autres fonctionnalités, qui n'ont pas pu être mise en place par manque de temps; comme une interface permettant une analyse quantitative et statistique des textes de la base, qui aurait pu être exportée sous la même forme que le rapport de résultats.

6. CONCLUSION

Ce stage de trois mois au croisement de plusieurs domaines (programmation, recherche, linguistique et fouille de textes) à l'étranger fût très stimulant pédagogiquement, culturellement et théoriquement. J'ai pu lors des diverses rencontres confronter mes idées reçues et obtenir des réponses aux questions que je me posais au sujet de mon avenir professionnel. Celui-ci m'aura aussi permis de découvrir une nouvelle structure de travail enrichissante, ainsi qu'une nouvelle vision de l'outil informatique et statistique plus littéraire.

J'ai eu la chance de gérer en autonomie un projet de A à Z, de sa conception à sa réalisation, en respectant un cahier des charges et une contrainte de temps. Cette expérience m'aura familiarisé avec la programmation orientée objets en me permettant de découvrir de nouvelles techniques que je n'avais pas encore eu l'occasion d'utiliser : la programmation événementielle, la mise en place d'une base de données embarquée, le modèle entité objet, l'utilisation de regex…

Je fus de plus très heureux d'avoir pu apporter une solution fonctionnelle à l'échéance de celui-ci, permettant ainsi d'offrir un outil personnalisé à M. Purnelle afin qu'il puisse poursuivre le travail de recherche de poèmes inédits avec les fonds de François Jacqmin. La documentation et le code commenté fournis permettront, je l'espère de maintenir et partager ce programme.

Pour ma part, ce projet m'aura fixé sur plusieurs aspects professionnels. En effet, cela m'aura donné un point de vue interne de la vie au sein d'une université, mais je ne me vois pas pour autant devenir développeur dans ce genre de structure, métier trop solitaire à mon goût. La littérature que j'ai pu découvrir par le biais du LASLA m'a par contre motivé à continuer à me former sur les techniques de traitement et de fouille de données textuelles et a bien entendu fait grandir l'attrait que je porte au monde de la recherche.

De par ce rapport j'espère vous avoir permis d'entrevoir comment la poésie se transpose en chiffres.

Rapport de stage

- Développement d'un outil d'analyse de poèmes -

De la poésie des chiffres aux chiffres de la poésie…

UNIVERSITÉ LUMIÈRE LYON 2
UNIVERSITÉ DE LYON

icom
INSTITUT DE LA COMMUNICATION

Département d'Informatique et de Statistique
UNIVERSITÉ LUMIÈRE LYON 2
DIS
Institut de la Communication

TABLE DES MATIERES

A. Format des fichiers txt d'input....1
B. Script r de mise en forme des fichiers originaux du corpus....3
C. Script sql de generation de corpus db....5
D. Modele entite-relation de la base de donnees fichier sql de generation de la base de donnees....6
E. Diagramme uml des classes associes a la base de donnees....6
F. Architecture de JPA....7
G. Gestion des objets de la base via JPA....7
H. Methode de calcul des indicateurs instancies....8
I. Diagramme uml des classes associees au requetes....10
J. Cas d'utilisation du gestionnaire de base de donnees....11
K. Cas d'utilisation de l'interface de creation de requetes....11
L. Cas d'utilisation de l'interface de resultats....12
M. ALGORITHME 1 : Création d'un poème à partir d'une chaine structurée....12
N. ALGORITHME 2 : Création d'un recueil à partir d'une chaine structurée....13
O. ALGORITHME 3 : Insertion des entités dans la base de données en cascade....14
P. ALGORITHME 4 : Création automatique d'un Anti-dictionnaire....15
Q. ALGORITHME 5 : Fonctions de formatage de texte de la classe MiseEnForme....16
R. Structure du template HTML dédié à la génération de rapports....18

A. FORMAT DES FICHIERS TXT D'INPUT

Structure d'un poème :

Un poème formaté contient obligatoirement une balise de titre (ligne 1) annonçant le début du poème. Entre la balise ouvrante et fermante peut être précisé, si le poème en possède, son titre (dans notre exemple le poème n'a pas de titre). L'ID précédé du symbole « $ » représente son référencement dans son recueil d'appartenance. Les lignes 2 à 9 sont quant à elles le contenu du poème : il est obligatoire.

```
1- <Titre_poeme id=$1></Titre_poeme>
2- Les lilas, les nerfs
3- la main les touche ;
4- la maison dans le poing
5- serre les vieux habits.
6- À présent, l'embellie,
7- la jambe exacte.
8- Et tu respires
9- sans y penser.
```

Structure d'un recueil :

Un recueil au format txt est composé obligatoirement d'un titre entre balise (ligne 2), ainsi que de poème(s) : dans notre exemple nous avons 3 poèmes. Les poèmes y sont alors formatés de la même manière que les fichiers ne contenant qu'un seul poème : avec une balise de titre (ligne 3, 13 et 21) indiquant le début de ceux-ci, dans lequel figure son id précédé d'un signe « $ », et un contenu (de la ligne 4 à 12 pour le premier poème). Le titre encore une fois peut ne pas être précisé.

```
1- <Titre_recueil>Six poemes</Titre_recueil>
2- <Titre_poeme id=$1>null</Titre_poeme>
3- Toucher.
4- Envenime l'épiderme ou le bossu,
5- venin malin des aulnes.
6- Bohémien.
7- Bleue, l'esquirre.
8- Le rite ensanglanté
9- que le duc dame.
10- Botte ou suture
11- ou mec au fronton.
12- <Titre_poeme id=$2>null</Titre_poeme>
13- Qui casse l'eau, casse l'aine
14- ou le héron tendu qui grogne
15- ou devient fétu, feu, fille.
16- De quel tandem, de quel idem
17- garder les noeuds ?
18- Noyés sont les conspirateurs
19- dès qu'on sonne l'alarme.
20- <Titre_poeme id=$3>null</Titre_poeme>
21- Bleu balbutieur sans enclume,
```

B. SCRIPT R DE MISE EN FORME DES FICHIERS ORIGINAUX DU CORPUS

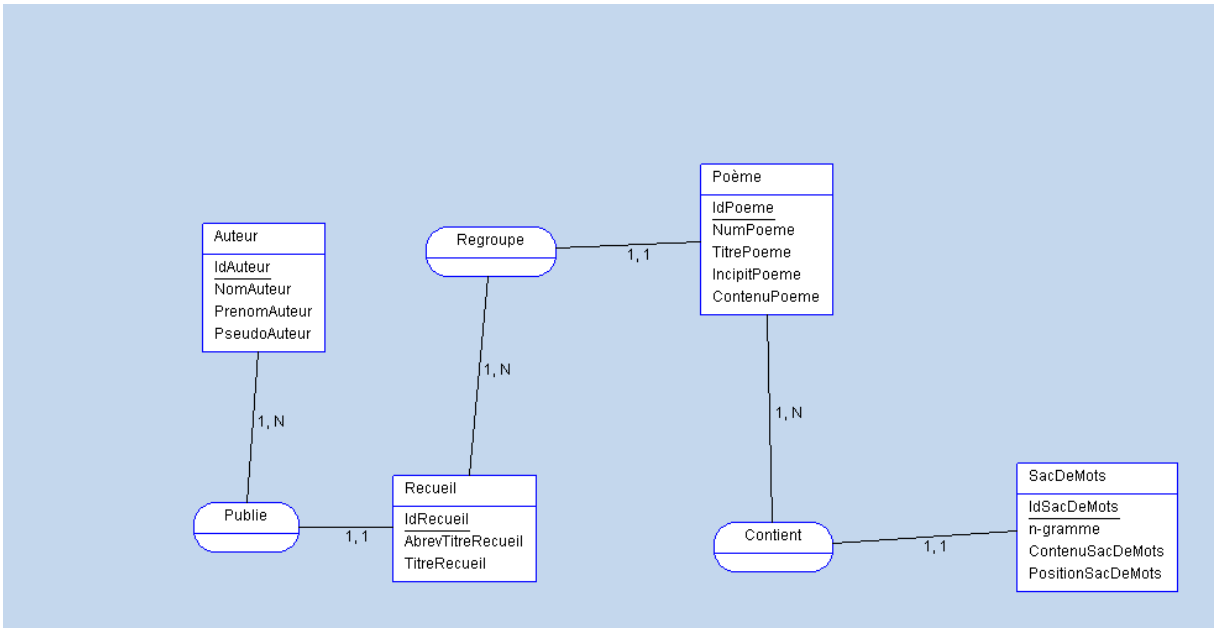
```
1-  #Chargement des fichiers d'index
2-  Index_poeme=read.csv("Poemes.csv",header = TRUE, sep=";")
3-  Index_recceuil=read.csv("Recueils.csv",header = TRUE, sep=";")
4-
5-  #Création des variables de chemins
6-  path_current_dir<-getwd()
7-  path_data_dir<-paste(path_current_dir,"/Izoard",sep="")
8-  path_data_dir
9-  setwd(path_current_dir)
10- getwd()
11- new_dir="Izoard_Balise"
12-
13- #Création du répertoire de dépot si non existant
14- #dir.create(new_dir)
15- path_new_dir=paste(path_current_dir,"/Izoard_Balise",sep="")
16-
17- #Création de la liste des poèmes
18- liste_poeme=dir(path_data_dir)
19- liste_poeme
20- liste_poeme_indice=matrix(data=NA, nrow=length(liste_poeme), ncol=2)
21-
22- #Création d'une matrice d'indexation
23- for(i in 1:length(liste_poeme)){
24-             liste_poeme_indice[i,1]=liste_poeme[i]
25-              liste_poeme_indice[i,2]=substr(liste_poeme[i], 0, 2)
26- }
27-
28- #Suppression des "0" devant les nombres inférieurs ? 10
29- for(i in 1:9){
30-             liste_poeme_indice[i,2]=substr(liste_poeme_indice[i,2],2,2)
31- }
32- liste_poeme_indice
33-
34- #PROCEDURE DE NETTOYAGE
35- flag=0 #initiation du flag de changement de dossier courant
36- setwd(path_data_dir)
```

```
for(i in 1:length(liste_poeme)){
  #Si déjà parcouru une fois
  if(flag==1){setwd(path_data_dir)}
  #lecture fichier
  str_temp <- readLines(con <- file(liste_poeme[i]))
  close(con)

  str_temp=sub("@","",str_temp)
  dol2=grep("\\$",str_temp
#id qui ne sont pas seul sur une ligne
  for(l in 1:length(dol2)){
    if(substr(str_temp[dol2][l],1,1)!="$"){
      ID_error=regexpr("\\$",str_temp[dol2][l])[1]
      ID_extract=substr(str_temp[dol2][l],ID_error,nchar(str_temp[dol2][l]))
      str_temp=c(str_temp[1:(dol2[l]-1)], substr(str_temp[dol2[l]],0,(ID_error-1)), ID_extract,
str_temp[(dol2[l]+1):length(str_temp)])
    }
  }

  #Suppresion des marqueurs de vers
  length_temp=nrow(str_temp)#Nb lignes

  #Reperage des titres existants
  indice_ok=which(Index_poeme[,1]==liste_poeme_indice[i,2])
  Num_po_indice_ok=Index_poeme[indice_ok,4]
  Num_po_indice_ok=paste("$",Num_po_indice_ok,sep="")

  #Reperage des balises de début poèmes
  indice_marquage_poeme=grep("\\$",str_temp)

  #Insertion des titres existants
for(j in 1:length(indice_ok)){
    test=which(Num_po_indice_ok==str_temp[indice_marquage_poeme[j]])
    if(length(test)!=0){

      tempo=str_temp[indice_marquage_poeme[j]]
      titre=Index_poeme[test,5]

      str_temp[indice_marquage_poeme[j]]=paste("<Titre_poeme
id=",tempo,">",titre,"</Titre_poeme>",sep="")
    }
  }

  #Insertion de titre vides pour poèmes non références
  sans_titre=setdiff(grep("\\$",str_temp),grep("=\\$",str_temp))
  str_temp[sans_titre]=paste("<Titre_poeme
id=",str_temp[sans_titre],"></Titre_poeme>",sep="")

  #Insertion du titre du receuil
titre=paste("<Titre_recueil>",Index_recceuil[liste_poeme_indice[i,2],3],"</Titre_recueil>",sep="")
  str_temp=c(titre,str_temp)
  str_tem=str_temp[-which(nchar(str_temp)==0)]
  setwd(path_new_dir)

  #Création du nouveau fichier
  con <- file(liste_poeme[i], open = "w")
  vect <- str_temp
  writeLines(vect, con = con)
  close(con)
  flag=1
}
```

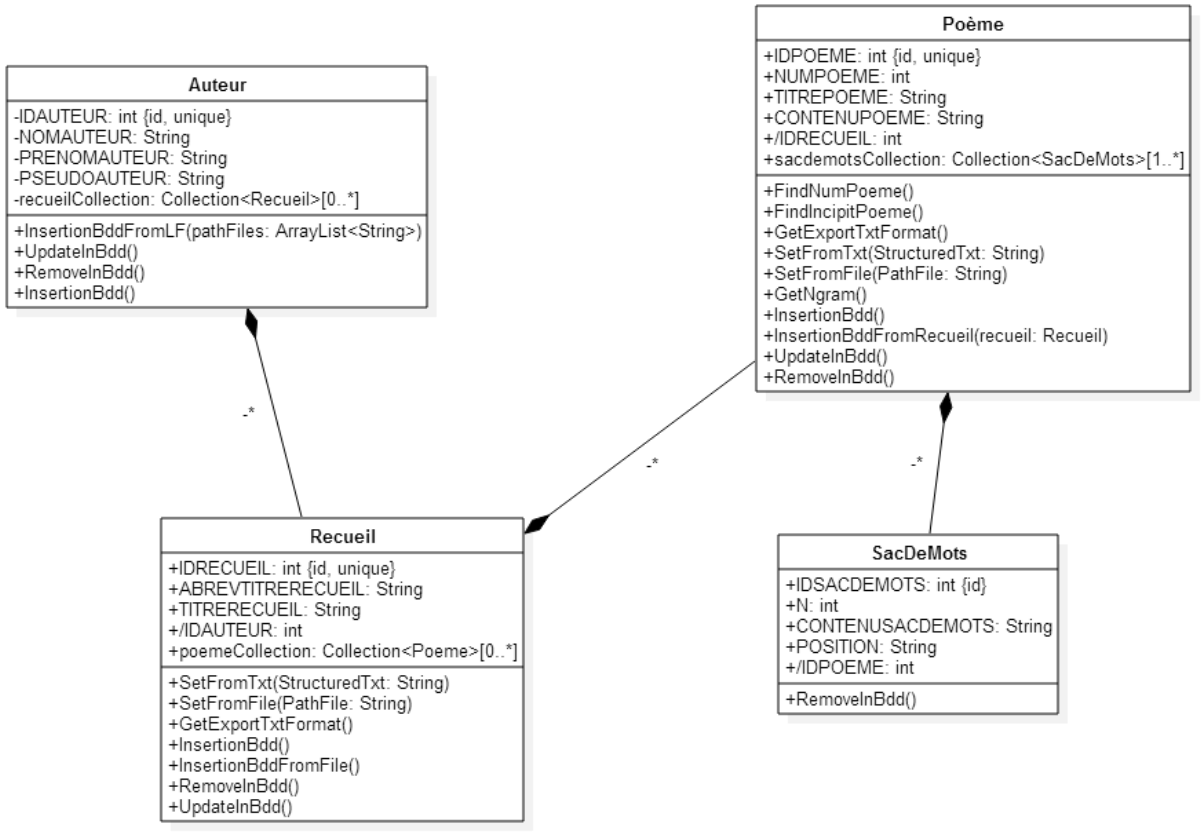
C. SCRIPT SQL DE GENERATION DE CORPUS DB

```
1-  DROP TABLE SacDeMots;
2-  DROP TABLE Poeme ;
3-  DROP TABLE Receuil ;
4-  DROP TABLE Auteur ;
5-
6-  CREATE TABLE Auteur (
7-  idAuteur INT NOT NULL GENERATED ALWAYS AS IDENTITY,
8-  NomAuteur VARCHAR(100),
9-  PrenomAuteur VARCHAR(100),
10- PseudoAuteur VARCHAR(100),
11- PRIMARY KEY (idAuteur) );
12-
13- CREATE TABLE Receuil (
14- idReceuil INT NOT NULL GENERATED ALWAYS AS IDENTITY,
15- AbrevTitreReceuil VARCHAR(250),
16- TitreReceuil VARCHAR(255),
17- idAuteur INT NOT NULL,
18- PRIMARY KEY (idReceuil) );
19-
20- CREATE TABLE Poeme (
21- idPoeme INT NOT NULL GENERATED ALWAYS AS IDENTITY,
22- NumPoeme INT,
23- TitrePoeme VARCHAR(250),
24- IncipitPoeme VARCHAR(250),
25- ContenuPoeme LONG VARCHAR,
26- idReceuil INT NOT NULL,
27- PRIMARY KEY (idPoeme) ) ;
28-
29- CREATE TABLE SacDeMots (
30- IdSacDeMots INT NOT NULL GENERATED ALWAYS AS IDENTITY,
31- N INT,
32- ContenuSacDeMots VARCHAR(255),
33- position INT,
34- idPoeme INT NOT NULL,
35- PRIMARY KEY (IdSacDeMots) );
36-
37-
38- ALTER TABLE Receuil ADD CONSTRAINT FK_Receuil_idAuteur FOREIGN KEY (idAuteur)
    REFERENCES Auteur (idAuteur);
39- ALTER TABLE SacDeMots ADD CONSTRAINT FK_SacDeMots_idPoeme FOREIGN KEY (idPoeme)
    REFERENCES Poeme (idPoeme);
40- ALTER TABLE Poeme ADD CONSTRAINT FK_Poeme_idReceuil FOREIGN KEY (idReceuil)
    REFERENCES Receuil (idReceuil);
41-
42- CREATE INDEX auteur_index
43- ON AUTEUR ( PSEUDOAUTEUR DESC , PRENOMAUTEUR DESC, NOMAUTEUR DESC);
44- CREATE INDEX poeme_index
45- ON POEME (NUMPOEME ASC, TITREPOEME ASC);
46- CREATE INDEX receuil_index
47- ON RECEUIL (ABREVTITRERECEUIL ASC, TITRERECEUIL ASC);
48- CREATE INDEX sdm_index
49- ON SACDEMOTS (idPoeme ASC,IdSacDeMots ASC,N ASC, POSITION ASC, CONTENUSACDEMOTS
    ASC);
50- CREATE INDEX sdm_index2
51- ON SACDEMOTS (idPoeme ASC,IdSacDeMots ASC,N ASC, POSITION ASC, CONTENUSACDEMOTS
    DESC);
```

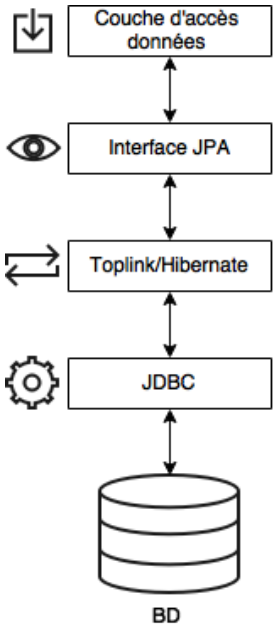
D. MODELE ENTITE-RELATION DE LA BASE DE DONNEES FICHIER SQL DE GENERATION DE LA BASE DE DONNEES



E. DIAGRAMME UML DES CLASSES ASSOCIES A LA BASE DE DONNEES

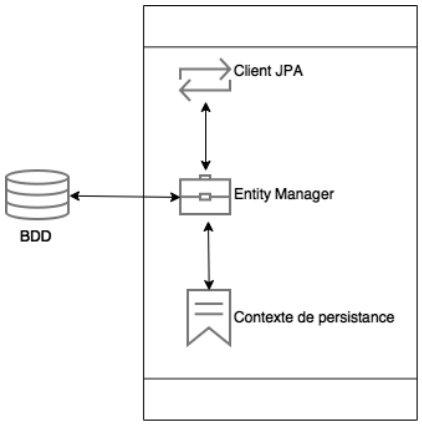


F. ARCHITECTURE DE JPA



La librairie JPA est composée de 4 couches, la première permettant l'accès à la base de donnée, la seconde faisant le lien entre la base de données et le modèle objet, troisième sont les classes d'objet de JPA, et la dernière permet d'accéder aux données les requête JPQL.

G. GESTION DES OBJETS DE LA BASE VIA JPA



Le logiciel passe donc par le manageur d'entité à chaque fois qu'une entité de la base est appelée. Un fichier xml lié au programme rappelle les règles de connexion et liste les classes d'accès aux données.

H. METHODE DE CALCUL DES INDICATEURS INSTANCIES

- Nombre d'occurrences distinctes :

$Nb\ Occ\ (P_i)_{distinctes}$ est le nombre de termes apparaissant dans le poème courant et la requête.

- Nombre d'occurrences :

$$Nb\ Occ(P_i) = \sum_{Nb\ Occ\ (P_i)_{distinctes}}^{j=1} f(t_j) \quad \{t_j \in requête, et\ t_j\ \in P_i)\}$$

Où P_i est le poème courant, $f(t_j)$ la fréquence documentaire du terme t_j appartenant à la requête et au poème.

- Indice de ressemblance :

$$\frac{Nb\ Occ(P_i)}{Nb\ mots(Pi)} \quad \{t_j \in requête, et\ t_j\ \in P_i)\}$$

- Indice d'efficacité :

$$\frac{Nb\ Occ\ Distinctes\ (P_i)}{Nb\ mots(requête)} \quad \{t_j \in requête, et\ t_j\ \in P_i)\}$$

- Distance inter-occurrence :

$$d(t_i, t_{i+1}) = |Pos(t_{i+1}) - Pos(t_i)| \quad \{t_i, t_{i+1} \in requête, et\ t_i, t_{i+1} \in P_i)\}$$

Où $Pos(t_i)$ est la position du terme dans le poème

- Distance moyenne :

$$d(Pi)_{moyenne} = \frac{\sum_{Nb\ occ-1}^{i=1} d(t_i, t_{i+1})}{Nb\ occ - 1}$$

- Distance minimale :

$$d(Pi)_{minimale} = Arg\ \min(|Pos(t_{i+1}) - Pos(t_i)|)$$

- Distance maximale :

$$d_{(Pi)maximale} = Arg\max(|Pos(t_{i+1}) - Pos(t_i)|)$$

- Le score Tf-Idf :

$$w_{TF-IDF}(t,d) = tf(t,d) * \log(N/df(t))$$

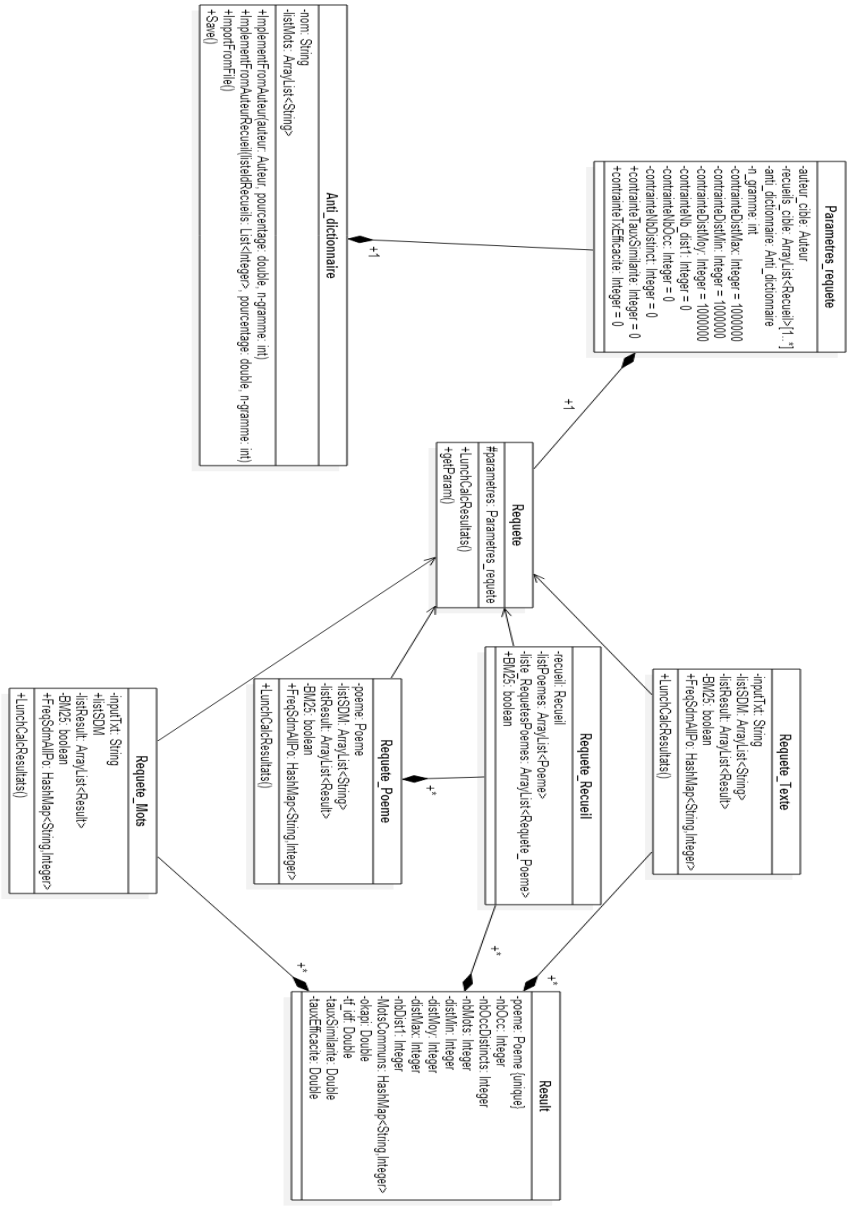
Où tf(t,d) correspond à la fréquence d'apparition du terme (soit le nombre d'occurrences présentes dans le poème t), et l'idf (log(N/df(t))) correspond au poids du terme considéré dans tout le corpus : N étant le nombre de documents et df(t) le nombre de poèmes dans lequel le terme t apparait. Pour un poème donné son tf-idf correspond donc à la somme de tous les tf-idf des termes le composant.

- Le score okapi :

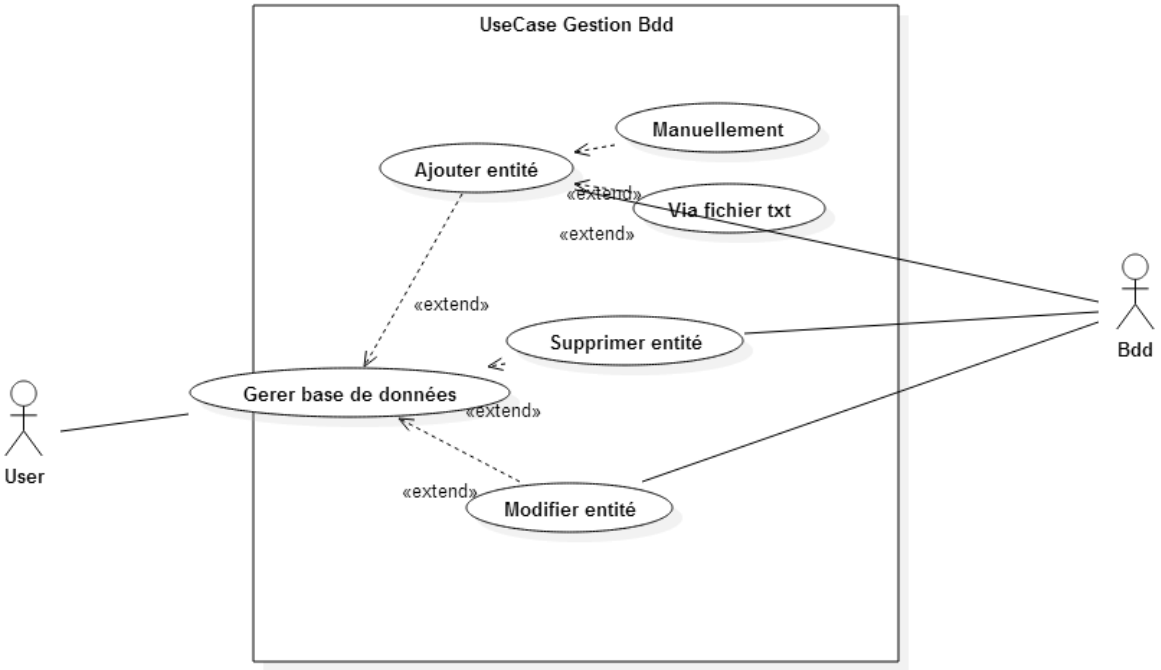
$$\begin{aligned} w_{BM25}(t,d) &= TF_{BM25}(t,d) * IDF_{BM25}(t) \\ &= \frac{tf(t,d) * (k_1+1)}{tf(t,d) + k_1 * (1 - b + b * dl(d)/dl_{avg})} * \log \frac{N - df(t) + 0.5}{df(t) + 0.5} \end{aligned},$$

Où k1=2 et b=0.75 sont des constantes, dl(d) la longueur du poème considérée, dlavg la moyenne des poèmes du champ de recherche, N le nombre de documents. Le score Okapi d'un poème donné correspond donc à la somme de tous les scores okapi des termes le composant.

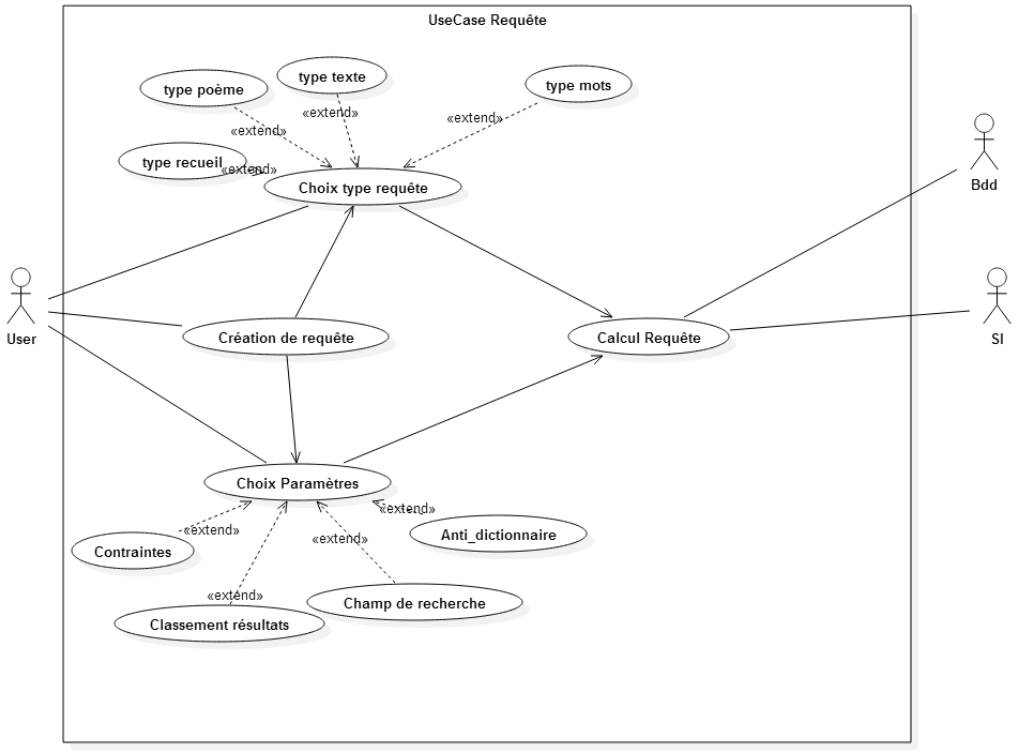
I. DIAGRAMME UML DES CLASSES ASSOCIEES AU REQUETES



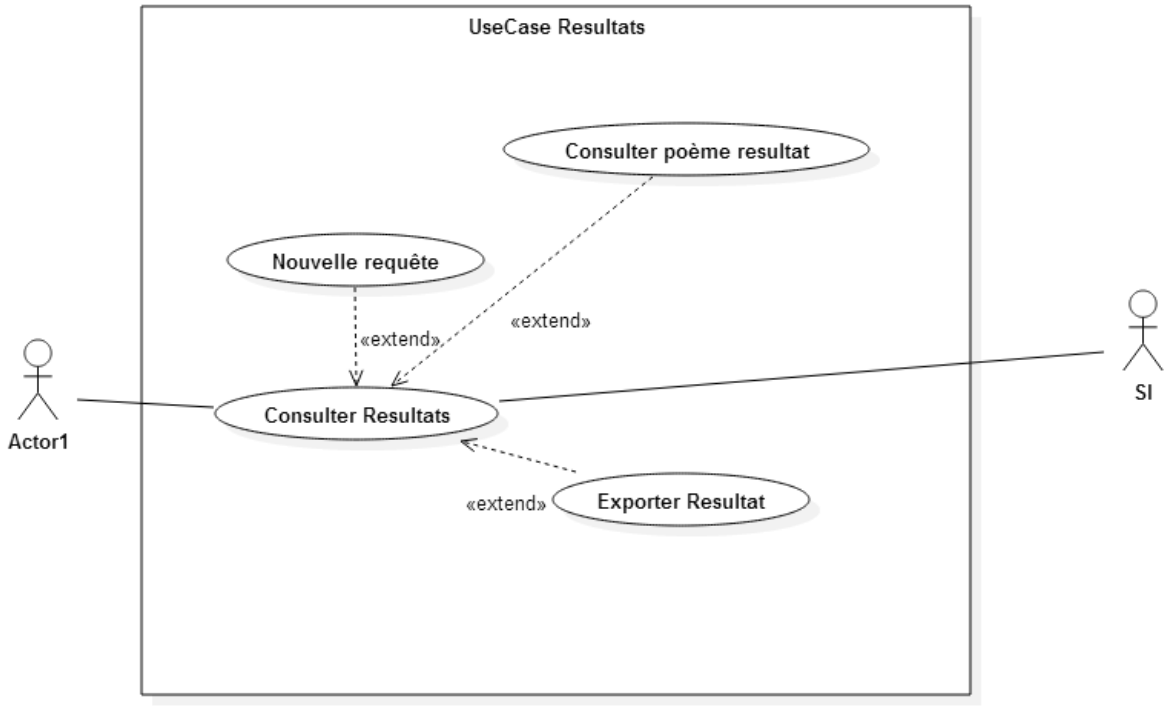
J. CAS D'UTILISATION DU GESTIONNAIRE DE BASE DE DONNEES



K. CAS D'UTILISATION DE L'INTERFACE DE CREATION DE REQUETES



L. CAS D'UTILISATION DE L'INTERFACE DE RESULTATS



M. ALGORITHME 1 : Création d'un poème à partir d'une chaine structurée

Méthode SetFromTxt() de la classe Poème

```
1-    public void SetFromTxt(String txt){
2-          //Recherche le pattern de titre afin d'en extraire le nom
3-          Pattern p_Titre = Pattern.compile(">(.*)<");
4-          Matcher m_Titre = p_Titre.matcher(txt);
5-
6-          //Si le titre est définit, implémenter celui du poème sinon = null
7-          if (m_Titre.find() && m_Titre.group(1).length()>1){
8-            this.setTitrepoeme(m_Titre.group(1));
9-          }else{
10-            this.setTitrepoeme(null);
11-         }
12-         //Extrait contenu du poème
13-         String pat_finTitre="</Titre_poeme>"+System.getProperty("line.separator");
14-         int begin=txt.indexOf(pat_finTitre)+pat_finTitre.length();
15-         int end=txt.length();
16-         this.setContenupoeme(txt.substring(begin, end));
17-         this.setIncipitpoeme(this.findIncipitpoeme());
18-      }
```

N. ALGORITHME 2 : Création d'un recueil à partir d'une chaine structurée

Méthode SetFromTxt() de la classe Recueil

```
1-    public ArrayList<Poeme> setFromTxt(String txt){
2-          //Trouve et implemente le titre
3-          String marqueur_debuttitre="<Titre_recueil>";
4-          String marqueur_fintitre="</Titre_recueil>";
5-          int debut=txt.indexOf(marqueur_debuttitre);
6-          int fin =txt.indexOf(marqueur_fintitre);
7-          this.setTitrereceuil(txt.substring(debut+marqueur_debuttitre.length(),fin));
8-
9-          ArrayList <int[]> Balise=new ArrayList <>();
10-         boolean encore=true;
11-         int k=1;
12-
13-         //Repère le debut et la fin de chaque poème
14-          while(encore==true){
15-             int tempo[]=new int[]{0,0};
16-             String pat="<Titre_poeme id=\\Q$"+k+">\\E(.*)</Titre_poeme>";
17-             String pat_next="<Titre_poeme id=\\Q$"+(k+1)+">\\E(.*)</Titre_poeme>";
18-             Pattern p = Pattern.compile(pat);
19-             Matcher m = p.matcher(txt);
20-             Pattern p_next = Pattern.compile(pat_next);
21-             Matcher m_next = p_next.matcher(txt);
22-
23-             if (m.find()){
24-
25-                tempo[0]=m.start();
26-                tempo[1]=m.end();
27-             }
28-
29-             if(m_next.find()==false)          encore=false; //if last tags -->stop
30-             Balise.add(tempo);
31-             k=k+1;
32-          }
33-
34-         //Extrait la liste de poèmes
35-         String tempo2;
36-         Poeme po;
37-         int index = 0 ;
38-         int begin;
39-         int end;
40-
41-         ArrayList<Poeme> ListePoReceuil=new ArrayList<>();
42-             while (Balise.size()-1>= index) {
43-              begin=Balise.get(index)[0];
44-              if(index!=Balise.size()-1){
45-              end=Balise.get(index+1)[0];
46-
47-              }else{
48-                     end=txt.length();
49-              }
50-              //Création d'un nouveau poème
51-             po= new Poeme(index+1);
52-             //Extraction de son contenu et implementation
53-             tempo2=txt.substring(begin,end-1);
54-             po.SetFromTxt(tempo2);
55-             //Trouve le numéro d'appartenance au reccueil
56-             po.setNumpoeme(index+1);
57-             //Ajout dans la liste
58-             ListePoReceuil.add(po);
59-             index++ ;
60-         }
61-         return ListePoReceuil;
62-
63-      }
```

O. ALGORITHME 3 : Insertion des entités dans la base de données en cascade

Fonction d'insertion la classe Auteur

```
1-  public void InsertionBddFromLF(ArrayList<String>  chemins) throws IOException{
2-          //Open conn
3-          EntityManagerFactory emf = Persistence.createEntityManagerFactory("JavaApplication10PU");
4-          EntityManager em = emf.createEntityManager();
5-
6-          int index;
7-          ArrayList<Receuil> LR=new ArrayList();
8-          LR=new ArrayList();
9-          int i=1;
10-         // for each file
11-         for(String cheminlr:chemins){
12-
13-             //Generate DB index for a recceuil
14-             if(((Number) em.createQuery("SELECT COUNT(r.idreceuil) FROM Receuil r").getSingleResult()).intValue()==0){
15-                 index=1;
16-             }else{
17-                 index=((Number) em.createQuery("SELECT MAX(r.idreceuil) FROM Receuil r").getSingleResult()).intValue()+1;
18-             }
19-             //Create recceuil
20-             Receuil R_tempo=new Receuil(index);
21-             //Call specific method for extract contents
22-             R_tempo.InsertionBddFromFile(cheminlr, this);
23-             i++;
24-          }
25-
26-          //close conn
27-          em.close();
28-          emf.close();
29-      }
```

Fonction d'insertion la classe Recueil

```
1-      public void InsertionBddFromFile(String chemin, Auteur a) throws IOException{
2-          //Open conn
3-          EntityManagerFactory emf = Persistence.createEntityManagerFactory("JavaApplication10PU");
4-          EntityManager em = emf.createEntityManager();
5-          int index;
6-          //Definition de l'id du recueil
7-          if(((Number) em.createQuery("SELECT COUNT(r.idreceuil) FROM Receuil r").getSingleResult()).intValue()==0){
8-              index=1;}else{
9-              index=((Number) em.createQuery("SELECT MAX(r.idreceuil) FROM Receuil r").getSingleResult()).intValue()+1;
10-         }
11-
12-         Auteur A=em.find(Auteur.class, a.getIdauteur());
13-         em.close();
14-         emf.close();
15-         this.setIdreceuil(index);
16-         //recupère liste de poème
17-         ArrayList<Poeme> Lp=this.setFromFile(chemin);
18-         //attribut recueil à l'auteur
19-         a.getReceuilCollection().add(this);
20-         //attribut auteur au recueil
21-         this.setIdauteur(a);
22-         //met à jour collection de recueil de l'auteur
23-         a.UpdateInBdd();
24-         //Insère chaque poème
25-         for(Poeme p:Lp){
26-             p.InsertionBddFromReceuil(this);
27-         }
28-
29-     }
```

Fonction d'insertion la classe Poème :

```
1-      public void InsertionBddFromReceuil(Receuil r){
2-          this.setIdreceuil(r);
3-          r.getPoemeCollection().add(this);
4-          for(int j=1;j<=3;j++){
5-              ArrayList<String> ListSDC=this.GetNgramm(j);
6-              ListIterator<String> it = ListSDC.listIterator();
7-              int i=1;
8-              while (it.hasNext()) {
9-                  Sacdemots SDC=new Sacdemots();
10-                 SDC.setContenusacdemots(it.next());
11-                 SDC.setN(j);
12-                 SDC.setPosition(i);
13-                 SDC.setIdpoeme(this);
14-                 this.getSacdemotsCollection().add(SDC);
15-                 i++;
16-             }
17-
18-         }
19-         this.InsertionBdd(r);
20-     }
```

P. ALGORITHME 4 : Création automatique d'un Anti-dictionnaire

Fonction ImplementFromAuteur(Auteur a,double perncent,int n) de la classe Anti-dictionnaire

```
1-      public void ImplementFromAuteur(Auteur a,double percent,int n){
2-          //Ouverture de la connection
3-          EntityManagerFactory emf = Persistence.createEntityManagerFactory("JavaApplication10PU");
4-          EntityManager em = emf.createEntityManager();
5-
6-          //Recherche des mots vides et de leurs pourcentage d'apparitions
7-          List<String> r= em.createQuery("SELECT s.contenusacdemots FROM Sacdemots s  WHERE s.n=:n AND
        s.idpoeme.idreceuil.idauteur.idauteur=:ida GROUP BY s.contenusacdemots ORDER BY COUNT(s.contenusacdemots)
        DESC").setParameter("n",n).setParameter("ida",a.getIdauteur()).getResultList();
8-
9-          ArrayList<String> lfinal=new ArrayList<>();
10-
11-         //Ajout des mots à l'anti-dico jusqu'au pourcentage limite
12-         int fin=(int) Math.round(r.size()*percent);
13-         ArrayList<String> tempo=new ArrayList<>();
14-         for(int i=0;i<=fin-1;i++){
15-             lfinal.add(r.get(i));
16-         }
17-         this.setListMots(lfinal);
18-
19-         //Fermeture de la connection
20-         em.close();
21-         emf.close();
22-     }
```

Q. ALGORITHME 5 : Fonctions de formatage de texte de la classe MiseEnForme

Première méthode de mise en forme de la classe MiseEnForme :

```
1-  public String[] ReturnFormatedTxt1(){
2-
3-      //Cas ou une mise en forme propre est possible
4-          FormatedStr[1]=this.ChaineRequete;
5-          FormatedStr[0]=this.ChainePoeme;
6-          //Pour chacuns des mots communs
7-          for(String p:this.MotsCommuns){
8-
9-              int l=p.length(); //longueur de chacun des mots pour recherche de chaine
10-             int index = this.FormatedStr[1].toLowerCase().indexOf(p); //recherche première occurences dans la requête
11-             //tant que le pattern est trouvé dans la chaîne
12-             while(index >=0) {
13-                 //Si le pattern trouvé ne fait pas partie d'un autre mot (utilisation des règles typographiques)
14-                 if((index==0 || " ".equals(this.FormatedStr[1].substring(index-1, index))
15-                     || " ".equals(this.FormatedStr[1].substring(index-1, index))
16-                     || "'".equals(this.FormatedStr[1].substring(index-1, index))
17-                     || "-".equals(this.FormatedStr[1].substring(index-1, index))
18-                     || "(".equals(this.FormatedStr[1].substring(index-1, index))
19-                     || "{".equals(this.FormatedStr[1].substring(index-1, index))
20-                     || "[".equals(this.FormatedStr[1].substring(index-1, index))
21-                     || "\"".equals(this.FormatedStr[1].substring(index-1, index))
22-                     || "'".equals(this.FormatedStr[1].substring(index-1, index))
23-                      || "'".equals(this.FormatedStr[1].substring(index-1, index))
24-                     || System.getProperty("line.separator").equals(this.FormatedStr[1].substring(index-1, index))
25-                     || "\n".equals(this.FormatedStr[1].substring(index-1, index)))
26-                 && ((    index+l+1>=FormatedStr[1].length())
27-                     ||" ".equals(this.FormatedStr[1].substring(index+l, index+l+1))
28-                     ||" ".equals(this.FormatedStr[1].substring(index+l, index+l+1))
29-                     ||".".equals(this.FormatedStr[1].substring(index+l, index+l+1))
30-                     || ",".equals(this.FormatedStr[1].substring(index+l, index+l+1))
31-                     || ".".equals(this.FormatedStr[1].substring(index+l,index+l+1))
32-                     || "'".equals(this.FormatedStr[1].substring(index+l, index+l+1))
33-                     ||"\"".equals(this.FormatedStr[1].substring(index+l, index+l+1))
34-                     || ")".equals(this.FormatedStr[1].substring(index+l, index+l+1))
35-                     || "}".equals(this.FormatedStr[1].substring(index+l, index+l+1))
36-                     || "]".equals(this.FormatedStr[1].substring(index+l, index+l+1))
37-                     || "-".equals(this.FormatedStr[1].substring(index+l, index+l+1))
38-                     || "'".equals(this.FormatedStr[1].substring(index+l, index+l+1)))
39-                     ){
40-                     // L'on met en forme celui-ci
41-
42-                     this.FormatedStr[1]=this.FormatedStr[1].substring(0,index)+"<span
    style=\"background:#FAAC58;\">"+this.FormatedStr[1].substring(index,
    index+l)+"</span>"+this.FormatedStr[1].substring(index+l, this.FormatedStr[1].length());
43-
44-                 }
45-                 //itération de l'index
46-                 index = this.FormatedStr[1].toLowerCase().indexOf(p, index+1);
47-                }
48-             .
```

……. Même chose pour le texte de FormatedStr[0], celui de la requête

```
92-
93-
94-
95-         return FormatedStr;
96-     }
```

Seconde méthode de mise en forme de la classe MiseEnForme :

```
1-    public String[] ReturnFormatedTxt2(){
2-            FormatedStr[0]="";
3-            FormatedStr[1]="";
4-            String FormatedStr2="";
5-            String FormatedStr1="";
6-
7-            //Formate chaines sous forme tabulaire
8-            TransformTxt Ttxt1=new TransformTxt( this.ChaineRequete,1);
9-            ArrayList<String> Str1=Ttxt1.getListMots1();
10-           TransformTxt Ttxt2=new TransformTxt(this.ChainePoeme,1);
11-           ArrayList<String> Str2=Ttxt2.getListMots1();
12-
13-           //Parcous tableau de mots du Poème courant
14-           for(String p:Str2){
15-
16-              //Formate si compris dans le tableau de mots de la requête
17-              if(Str1.contains(p)){
18-                 FormatedStr1=FormatedStr1+" <span style=\"background:#FAAC58;\">"+p+"</span> ";
19-              }else{
20-                 FormatedStr1=FormatedStr1+" "+p;
21-              }
22-           }
23-
24-           //Parcous tableau de mots de la requete
25-           for(String p:Str1){
26-
27-              //Formate si compris dans le tableau de mots du poème courant
28-              if(Str2.contains(p)){
29-                 FormatedStr2=FormatedStr2+" <span style=\"background:#FAAC58;\">"+p+"</span> ";
30-              }else{
31-                 FormatedStr2=FormatedStr2+" "+p;
32-              }
33-           }
34-
35-           FormatedStr[0]=FormatedStr1;
36-           FormatedStr[1]=FormatedStr2;
37-           return FormatedStr;
38-       }
```

R. Structure du template HTML dédié à la génération de rapports

```
<!DOCTYPE HTML>
<html>

<head>
  <title>$title</title>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta name="description" content="Ce document est issu d'une requete sur corpus" />
  <meta name="keywords" content="Squelette,html,documentation" />
  <meta name="author" content="Marcellus Wallace" />
  <style>
        AVEC STYLE CSS DEFINI
</style>
</head>

<body>
        <header>
        <img id="logo" src="$cheminIMG"  >
                <h1>POESTRY MINER</h1>
                <h1>Rapport d'analyse</h1>

        </header>
        <span id="req">
                <h2> Requête  :</h2>
                <p>
                        date :          $date </br>
                        nombre de mots : $LENGTHQUERY </br>
                        nombre de poèmes total dans les recueils selectionnés : $NBPO </br>
                        nombre de poèmes possédant au moins un n-gramme en commun avant la requête : $nbres </br>
                        nombre de resultats exportés : $nbexport </br>

                </p>

                <p>$queryRecueil<p>
                <p>$queryPo<p>

                <h3> Contenu : </h3>
                <p id="Poeme">$contenuQuery</p>
        </span>
        <span id="champQuery">
        <h2> Champs de la recherche</h2>
                <ul>
                        <li>auteur : $auteur</li>
                        <li>recueils : </li>
                                $recueils
                </ul>
        </span>

        <span id="param" class="floating-box">
        <h2> Paramètres</h2>
                <ul>

                        <li>type requete : $type_requete</li>
                        <li>n-gramme étudié : $n_gram</li>
                        <li>classés selon : $classement </li>
                </ul>
        </span>
        <span id="contraintes" class="floating-box">
        <h2> Contraintes</h2>
                <ul>
                        <li>nombre d'occurences minimum : $nbocc     </li>
                        <li>nombre d'occurences distinctes minimum : $nbdistinct</li>
                        <li>nombre de distances égales à 1 minimum : $nbdist1</li>
                        <li>contrainte sur la distance minimum : $distmin</li>
                        <li>contrainte sur la distance moyenne : $distmoy</li>
                        <li>contrainte sur la distance moyenne : $distmax</li>
```

```
                    <li>contrainte sur le pourcentage d'occurences de la requêtes en fonction du nombre de mots du poème courant : $perc1</li>
                    <li>contrainte sur le pourcentage d'occurences distinctes des mots de la requête en fonction du nombre de mots différents dans la requête : $perc2</li>
                </ul>
            </span>
            <span id="antiD" class="floating-box">
            <h2> Antidictionnaire utilisé :</h2>
                <p>$antidic</p>
            </span>
<span class="after-box">
  <h1>TABLEAU DE RESULTATS.</h1>
 <table>
<tr>
<th>ID Poème</th><th>ID Recueil</th><th>Titre Recueil</th><th>Numéro</th><th>Titre Poème</th><th>Taux de similarité (%)</th><th>Taux d'efficacité(%)</th><th>Nombre d'occurences</th><th>Nombre d'occurences distinctes</th><th>Nombre de mots</th><th>Distance minimale inter-occurences</th><th>Distance moyenne inter-occurences</th><th>Distance maximale inter-occurences</th><th>Score Tf_Idf</th><th>Score Okapi BM25</th>
</tr>
$table_res_content
</table>
<h1>DETAILS DES RESULTATS.</h1>
<table>
<tr>
<th id="td1">ID poème</th><th id="td1">ID Recueil</th><th id="td2">Texte requête</th><th id="td2">Texte poème</th><th id="td2">Mots Communs</th>
</tr>
$table_compare
</table>
</span>
</body>

</html>
```

Vue de la maquette :