MASTER 2 SITIS

Projet INF203 Recherche d'information

Rapport

23/01/19

Ines BAGHOU
Thomas CHEUNG-TOI-CHEUNG
Charline CLAIN
Pauline GAFFEZ
Kevin OUAZZANI

Sommaire

Introduction

Ce travail s'appuie sur les librairies java de Lucene. L'objectif est de réaliser une application de Recherche d'Information (RI) permettant la recherche de termes médicaux dans un corpus d'articles scientifiques. Un mode d'emploi est disponible dans le fichier README inclus dans le ZIP.

Le scénario est le suivant :

- Les documents indexés sont des abstracts d'articles scientifiques issus de PubMed pour les articles anglais et de LiSSa pour les articles français
- Le clinicien possède une base de données contenant ses patients auxquels sont reliés des diagnostics codés en CIM-10 (DP, DR, DAS)
- Le clinicien saisit le numéro du patient pour lequel il souhaite avoir de la littérature d'intérêt, en rapport avec les pathologies diagnostiquées (CIM-10) du patient.
- Il choisit ensuite le diagnostic qui l'intéresse plus particulièrement parmi les diagnostics affichés
- Il choisit également s'il souhaite créer un index ou requêter un index existant
- L'utilisateur choisit ensuite la langue dans laquelle il souhaite effectuer la recherche et s'il souhaite effectuer une requête simple ou étendue
- L'application renvoie les documents pertinents, c'est-à-dire les abstracts du corpus qui répondent le mieux à la requête, au regard du diagnostic sélectionné.

Les rôles ont été les suivants :

- Interface graphique php : Charline
- Constitution des corpus et indexation : Charline, Thomas
- Searcher, extension de requête : Pauline, Inès
- Elastic Search et Kibana : Kevin
- Rédaction du rapport : toute l'équipe

I. Implémentation

a. Main

Le Main appelle les deux classes utilisées par l'application : l'Indexer et le Searcher. A partir d'un menu, l'utilisateur choisit s'il souhaite indexer des documents ou requêter sur un corpus existant en fonction de la langue choisie. L'utilisateur peut quitter l'application en appuyant sur la touche « q » à tout instant.

b. Classe TextFileIndexer

La classe d'indexation permet d'indexer un corpus existant ou de mettre à jour un index existant en traitant de nouveaux documents.

i. Corpus et documents indexés

Deux collections ont été créées pour ce travail : l'une en français et l'autre en anglais. Chaque document est au format .txt et est un abstract issu du site PubMed (documents en anglais) ou LiSSa (documents en français). A défaut d'avoir pu établir une connexion à PubMed ou LiSSa pour automatiser la création du corpus, il a été nécessaire de les créer manuellement après une recherche. PubMed permet d'accéder directement aux abstracts des articles. Quant à LiSSa, le résumé a été extrait à la main. Il n'y a pas de fonctionnalité permettant de générer des abstracts aux formats .xml ou .txt. Un corpus a donc été créé et enregistré localement pour chacune des deux langues.

Les documents sont issus d'une première recherche sur le moteur de recherche de PubMed ou de LiSSa : les termes saisis sont les libellés de la CIM10 en anglais dans le premier cas et en français dans le second, correspondant donc à des pathologies de cette classification. Au total, les corpus sont constitués de 13 et 11 abstracts respectivement en anglais et en français et portant sur différentes pathologies.

ii. Indexer

Pour chaque index à créer, la classe Indexer propose la séquence présentée dans la table 1.

Table 1 : Etapes de l'indexation et méthodes employées

	Etape	Méthodes utilisées
1	Saisie du chemin du répertoire dans lequel sera créé l'index.	CreationIndex()
2	Choix de la langue d'indexation (anglais ou français). *Cette distinction est importante puisque l'Analyzer de Lucene qui traite le document n'est pas le même en fonction de la langue choisie.*	CreationIndex() - IndexWriterConfig()
3	Saisie du chemin du répertoire contenant les documents au format .txt à partir desquels créer l'index. *Dans notre cas il s'agit des abstracts issus de PubMed ou LiSSa.* Cette étape boucle jusqu'à ce que l'utilisateur appuie sur la touche « q ».	CreationIndex() - InputStreamReader() - indexFileorDirectory()

Si un index existe déjà, il est possible de l'utiliser directement (choix laissé à l'utilisateur).

c. Classe Searcher

Pour rappel, une recherche se traduit par la comparaison d'une requête et d'un corpus (par le biais d'un index) après traitement automatique de la langue (avec un analyzer sous Lucène) La figure 1 ci-après schématise ce fonctionnement :

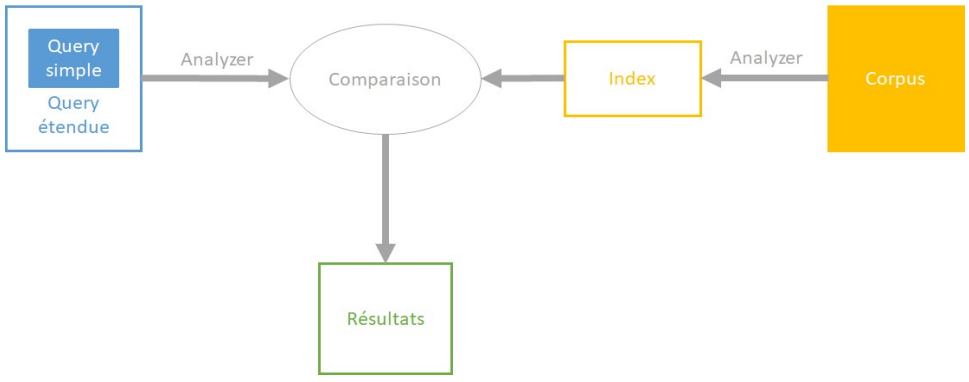

Figure 1 : Principe de la recherche d'information

Le rôle de cette classe est de rechercher dans l'index créé précédemment les termes relatifs au diagnostic du patient choisi. L'application propose ainsi à l'utilisateur un ensemble de documents pertinents sous la forme d'une liste avec un score de pertinence pour chaque document (tf-idf).

L'utilisateur peut effectuer soit :

- une recherche simple, dans ce cas la query est définie par le libellé CIM10 dans la langue choisie.
- une recherche étendue, dans ce cas la query est définie par l'ensemble des termes synonymes dans l'UMLS. Pour ce faire, le CUI du code CIM10 est récupéré via une table importée en local à partir de l'UMLS disponible sur le serveur Neptune. L'API UMLS est ensuite utilisée pour retourner tous les libellés des synonymes correspondants (correspondant aux atomes), qui complètent la query.

Le programme fonctionne à partir d'un switch dont chaque cas dépend des entrées clavier de l'utilisateur. Pour chaque recherche un Analyzer est instancié. En parallèle, un IndexSearcher est également instancié avec le chemin de l'index utilisé. Une query est envoyée pour chaque terme. Dans les cas des requêtes étendues, il y a donc envoi de plusieurs queries. Le résultat retourné indique le ou les documents renvoyés et le score proportionnel à la pertinence de la recherche.

d. Interface

Une interface a été implémentée sous PHP et est disponible dans le .zip rendu avec ce rapport (aperçu en figure 3). Elle communique avec la base de données MySQL fournie lors d'un projet précédent. Cette interface permet au clinicien :

- De saisir un numéro de patient (barre de saisie texte)
- D'afficher les codes CIM-10 correspondant aux diagnostics de ce patients
- De sélectionner les diagnostics d'intérêts pour lesquels il souhaite obtenir de la littérature sous forme d'abstracts
- De choisir d'effectuer une requête simple ou étendue (cf. expansion de requête) correspondant à ces diagnostics.

Figure 2 : Aperçu de la page d'accueil de l'interface PHP

II. Limites et perspectives

a. Evolution du schéma global de l'application

L'application devait initialement s'appuyer sur ElasticSearch pour l'indexation et la visualisation des résultats via Kibana. L'application devait utiliser l'API java d'ElasticSearch (figure 3). À cause des contraintes techniques et temporelles, ce modèle a été revu et simplifié (figure 4). Finalement, l'application repose sur un corpus fait à la main et indexé par Lucène. L'interrogation se fait directement par Lucène également et les résultats ne sont donc pas présentés par Kibana.

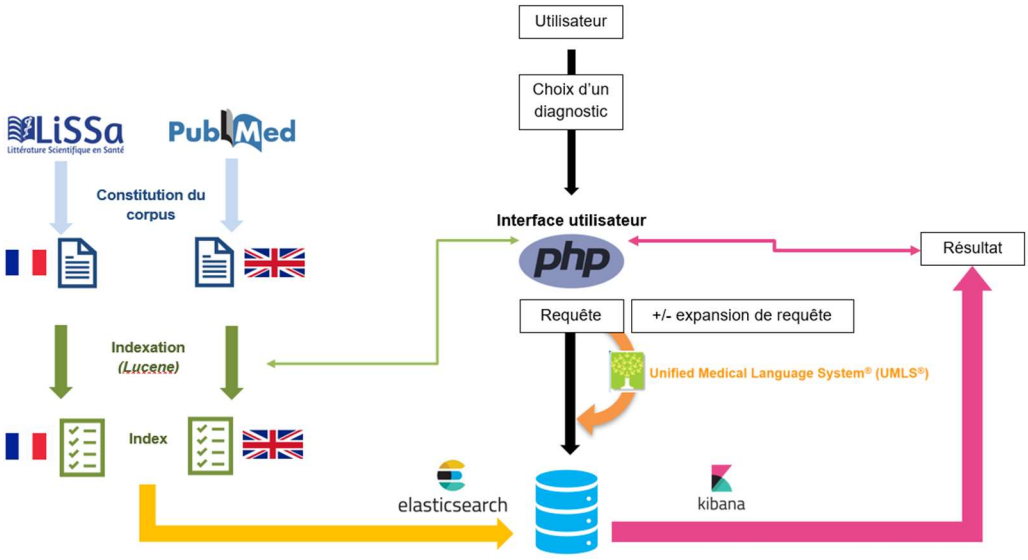

Figure 3 : Schéma global de l'application envisagé

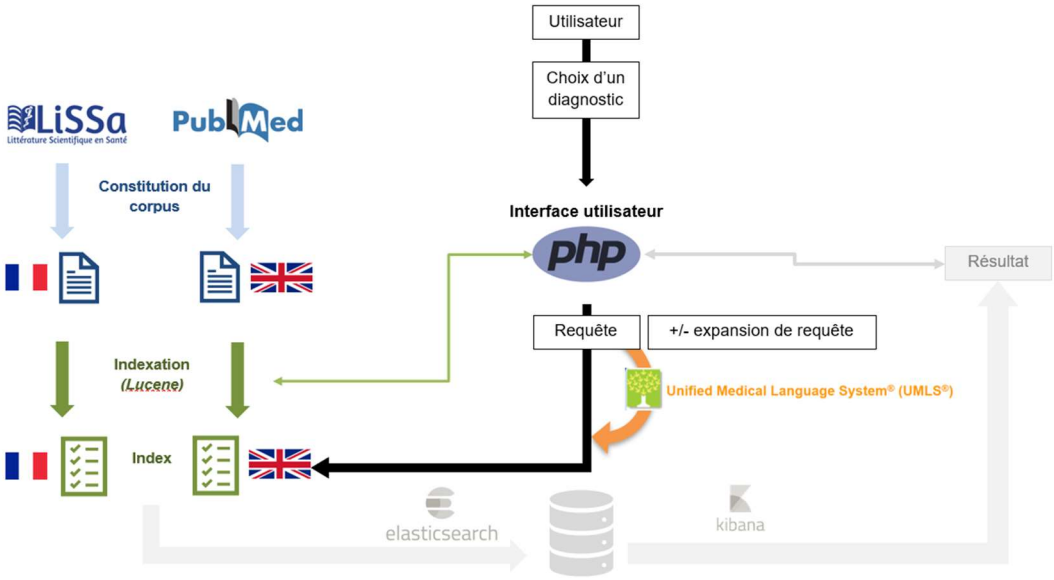

Figure 4 : Schéma global de l'application réalisé

b. Limites et perspectives de l'implémentation

Comme expliqué précédemment, le projet s'est heurté à des difficultés et le modèle idéal n'a pas pu être implémenté. Plusieurs limites et perspectives d'amélioration ont été identifiées :

- Constitution du corpus :

Une automatisation depuis PubMed vers un serveur Elasticsearch avait été envisagée, mais les contraintes techniques et temporelles n'ont pas permis de la mettre en place. Les corpus ont donc été constitués manuellement et sont plutôt restreints, ce qui ne permet pas une bonne évaluation de la pertinence de notre moteur de recherche.

- Interface :

Cette interface n'est pas complètement fonctionnelle. L'objectif à atteindre étant de l'intégrer totalement à l'application java et de faire en sorte que les choix de l'utilisateur communiquent simplement avec le programme. Une utilisation de servlets serait possible.

- ElasticSearch et Kibana :

Initialement, il était prévu que l'indexation des documents se fasse sur un cluster ElasticSearch. A l'aide des dépendances java « Jackson », les documents au format .txt étaient transformés en documents json puis envoyés sur le cluster de serveurs via l'API java d'ElasticSearch. L'ensemble des commandes de l'API REST de java permet d'effectuer des requêtes sur le cluster (GET, PUT, création et modification d'index…). Cette partie était théoriquement implémentée mais par manque de temps elle n'a pas été rajoutée au projet démo final. L'utilisation de Kibana aurait également permis d'obtenir une visualisation intéressante des documents sur une page web afin d'apporter du dynamisme à notre application. La présentation des résultats peut enfin être améliorée avec une présentation sous forme d'extraits des abstracts avec les termes reconnus mis en évidence dans les documents de manière visuelle (soulignés ou surlignés).

- Expansion de requête :

L'interface pourrait proposer une sélection manuelle des termes synonymes existants pour étendre la recherche, présentée sous forme de liste avec des éléments à conserver ou non. L'utilisateur serait ensuite libre de choisir le, ou les terme(s) qu'il souhaite envoyer pour effectuer la recherche. L'utilisation d'une table locale pour la sélection des termes CIM-10 pourrait également être revue au profit d'une API Java.

- Ranking

Il serait également envisageable de proposer un classement composite des documents prenant en compte l'impact factor (valeur scientifique) des revues dans lesquelles sont publiées les articles et le tf-idf (pertinence pure) pour le rendu des résultats.