



UNIVERSITÉ DE TOULOUSE III

RECHERCHE D'INFORMATIONS

Conception et implémentation d'un moteur de recherche

Auteurs :

Mme. Salima AZZOU
M. Max HALFORD
M. Kafil EL KHADIR
M. Rémy SIMON

Responsables :

Mme. Lynda LECHANI
Mme. Wahiba BAHOUN
Mme. Laure SOULIER
M. Bilel MOULAH



Table des matières

Remerciements	3
1 Présentation globale du document	4
1.1 Objet du document	4
1.2 Objectif du projet	4
2 Introduction	5
2.1 Cahier des charges	5
2.2 Objectifs	7
2.3 Méthodologie	7
3 Traitement des données	8
3.1 Analyse des données	8
3.2 Parsing	8
3.3 Indexation	8
3.4 Stockage	9
3.5 Appariement	10
4 Interface en ligne	12
4.1 Approche et objectifs	12
4.2 Solutions	12
4.3 Architecture du site	13
4.4 Requête par terme(s)	14
4.5 Requête avancée	16
4.6 Statistiques	18
4.7 Paramètres	20
4.8 Informations	20
5 Performance du SRI	21
5.1 Pertinence / Rappel	21
5.2 Temps de réponse	22
6 Bilan	23

6.1	Le produit	23
6.2	Propositions d'améliorations	23
6.3	L'équipe	23
6.3.1	Rémy Simon	23
6.3.2	Max Halford	24
6.3.3	Kafil El Khadir	24
6.3.4	Salima Azzou	24
Références		25
7	Annexes	26
7.1	MongoDB	26
7.2	LAMPP	27

Table des figures

1	Diagramme GANTT	6
2	Schéma de la base de données	10
3	Page d'accueil du site web	13
4	Exemple de requête par termes	14
5	Fenêtre modale pour commenter	15
6	Fenêtre modale d'informations	16
7	Exemple de requête avancée	17
8	Diagramme circulaire des documents les plus verbeux	18
9	Histogramme des lemmes les plus utilisés	19
10	Nuage de lemmes	19
11	Disposition de l'onglet "Paramètres"	20
12	Exemple de graphe des courbes de Rappel/Précision pour la méthode TFIDF	21
13	Exemple d'analyse des temps de réponse	22

Remerciements

Nous tenions à remercier Mme. Bahsoun pour les notions de gestion de projet qu'elle a su nous inculquer et qui nous a permis la réalisation des objectifs.

Nous souhaitons aussi à remercier Mme. Lechani pour son enseignement en Recherche d'Information, qui nous a été indispensable pour ce projet.

Nous remercions enfin également à remercier Mme. Soulier et M. Moulahi pour leur encadrement lors des différentes séances de travaux pratiques et dirigées ainsi que pour leurs disponibilités.

1 Présentation globale du document

Ce document a été rédigé par quatre étudiants de la L3 SID et il est destiné à l'Université Paul Sabatier. Il représente l'évolution et le fruit de notre travail dans le cadre d'un projet pour le second semestre de l'année Universitaire 2014/2015. L'objectif principal de ce projet, qui se situe dans le cadre de l'Unité d'Enseignement *Recherche d'information*, est de mettre en pratique les connaissances acquises tout au long de l'année dans les différents UE de la formation SID.

1.1 Objet du document

Ce rapport a pour but de rendre compte de l'ensemble de notre travail. On va donc y décrire toutes les démarches qui ont été adoptées pour la bonne réalisation de notre projet. A travers ce document, on va également décrire les différents logiciels qui ont été utilisés ainsi que les méthodes d'utilisation de ces logiciels.

1.2 Objectif du projet

L'objectif de ce projet était donc de mettre en pratique les connaissances acquises cette année au travers de la création d'un moteur de recherche d'information. Notre objectif final est que le moteur de recherche nous fournisse les documents les plus pertinents possibles en réponse à une requête soumise en entrée, ainsi que d'autres nombreuses informations sur ces documents

2 Introduction

Afin d'appliquer les méthodologies et les notions enseignées en premier semestre de la L3 SID Université Paul Sabatier, nous devons réaliser un Travail d'Etude et de Recherche durant 2 mois. Celui-ci nous permet à nous, étudiants, de nous initier à la recherche, d'appliquer les connaissances acquises durant notre cursus universitaire et de favoriser le travail en groupe encadré par des enseignants-chercheurs. Le projet que nous devons réaliser est un moteur de recherche. Afin de comprendre la démarche que nous avons utilisée pour mener ce projet à son terme, notre rapport se structure de la façon suivante : Tout d'abord, dans une première partie nous présenterons le cadre général de notre projet, c'est à dire les objectifs que nous avons voulu atteindre. Dans une seconde partie, nous introduirons le travail d'étude et de recherche que nous avons effectué. Dans une troisième partie, nous rentrerons dans un aspect plus technique en expliquant comment nous avons développé le projet avec ces différentes étapes. La quatrième partie sera vouée au développement de l'interface graphique. Nous consacrons une cinquième partie à l'utilisation de MongoDB car c'est un concept non vu cette année et que nous avons grandement utilisée. Nous ferons une sixième partie sur l'assurance et le contrôle qualité. Nous terminerons enfin par un bilan sur notre projet.

2.1 Cahier des charges

L'objectif principal du projet a été la mise en place d'un système de recherche d'information complet. Nous nous sommes donc lancés tous les 4 dans la création de ce travail. Pour parvenir à l'aboutissement de notre projet, nous sommes référencés aux enseignements suivis durant notre cursus, sans qui lesquels, notre réalisation finale n'aurait pas été possible. Voici un récapitulatif des compétences utilisées :

- Recherche d'Information
- Génie logiciel/Processus
- Langage de programmation (Perl)
- Conception de bases de données
- Langage de requêtes

Sur un point de vue pratique l'objectif principal sera suivi par le biais de la conception, l'analyse, l'implémentation et l'évaluation de ce système de recherche.

Organisation

1. Le travail sera organisé en groupe, de quatre personnes de préférence. Il est recommandé d'utiliser un chef de projet.
2. Tout le déroulement du projet se fera en plus grande partie durant les séances de travaux dirigés et travaux pratiques. Un travail personnel complémentaire nous est demandé afin de compléter le projet et de finaliser le rendu final. Des revues seront prévues afin de rendre compte du résultat de manière régulière.
3. Les tâches ont été réparties plutôt équitablement au travers chaque membre du groupe afin d'optimiser au mieux le temps de travail.
4. Il est prévu de rendre un rapport écrit à la fin du projet. Une soutenance orale sera également demandée afin d'appuyer le dossier écrit.

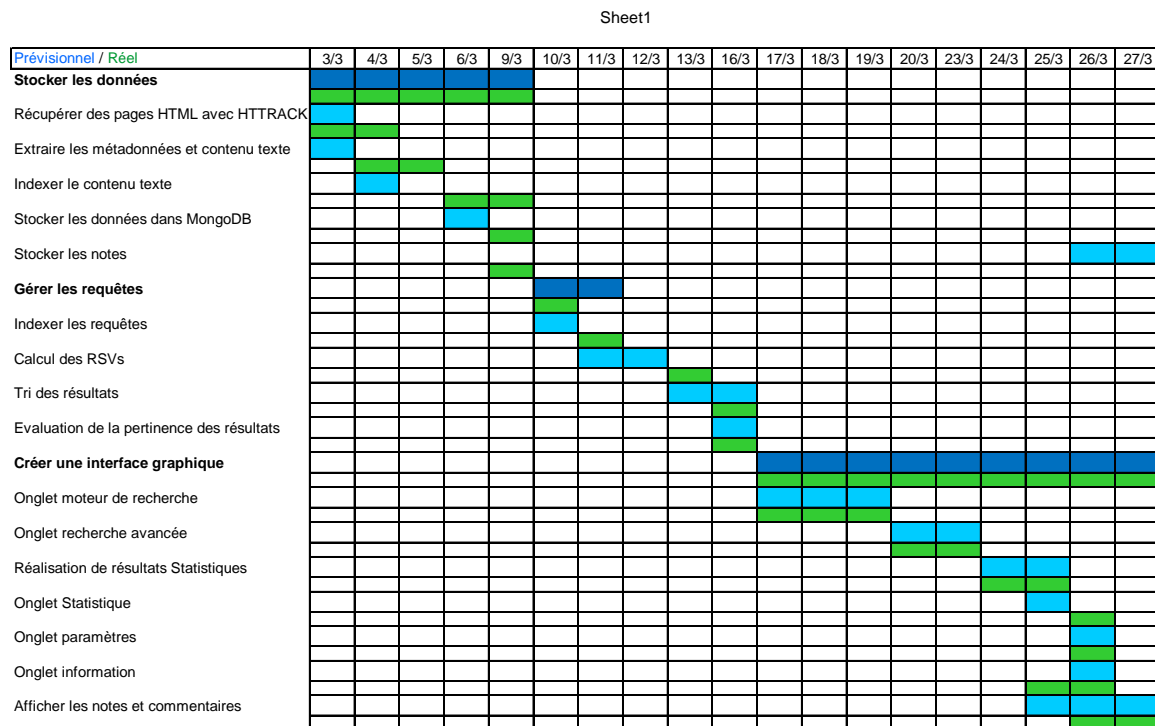


FIGURE 1 – Diagramme GANTT

Évaluation

La note finale sera basée essentiellement sur 3 critères :

- La qualité du travail réalisé
- La qualité du support écrit
- La qualité de la soutenance : supports, répartition du temps de parole, réponses aux questions posées

Le dossier final rendu sera noté et comptera pour 40% de la moyenne finale de l'enseignement "Concepts RI".

Travail à faire

Les différentes étapes concernant la mise en oeuvre du système de recherche d'information ont donc été les suivantes :

Étape 1 : Préparation de la collection

Étape 2 : Traitement des requêtes

Étape 3 : Évaluation des performances de recherche

Étape 4 : Mise en oeuvre d'une interface de recherche

2.2 Objectifs

L'objectif final est donc de créer un logiciel capable de sélectionner, à partir d'une collection de documents, une liste de ceux qui seraient pertinents en réponse à une requête utilisateur. Il s'agirait également d'évaluer les performances de recherche du système initialisé.

2.3 Méthodologie

Cette partie est destinée à rendre compte de la marche que nous avons suivie tout au long du déroulement de ce projet. En effet contrairement aux autres groupes nous avons décidé, après concertation de tous nos membres, d'adopter une démarche différente. Nous avons donc utilisé le système de gestion de base de données MongoDB qui a la particularité de faire partie de la mouvance NoSQL. Ce pari était à la fois risqué, car nous nous isolions des autres groupes et prenions donc à notre charge un travail naturellement plus conséquent. Mais également valorisant pour nous car nous ne dépendions plus des autres groupes et étions beaucoup plus indépendant, ce qui pouvait se révéler être un atout pour notre groupe. Tout comme le traitement des données, nous avons fait le choix de créer notre propre interface graphique. Là aussi nous sommes restés autonomes et avons élaboré cette interface web propre à notre groupe.

3 Traitement des données

3.1 Analyse des données

Les données que l'on a traité sont des pages webs provenant de site d'informations *Wikinews*. Il se trouve que l'on avait eu affaire à ces données lors du S5 pour un autre projet. Les données ne sont pas très complexes, les métadonnées à récupérer sont :

- La date d'écriture
- La dernière date de modification
- Les sources sous forme d'URLs
- Les catégories

On peut remarquer qu'une page web peut contenir plusieurs sources et plusieurs catégories. Cette propriété anodine peut inutilement complexifier une base de données supportant seulement des valeurs atomiques. Nous expliquons dans la partie 3.4 comment nous avons contourné ce problème.

3.2 Parsing

Nous avons décidé de récupérer les données "à la main". En effet, il ne nous a fallu que 15 minutes pour télécharger les pages une par une. Ecrire un script nous aurait pris plus de temps. N'étant que 4 cette décision nous a fait gagner du temps non négligeable par rapport aux autres groupes. Comme dit précédemment nous avons déjà traité des données de Wikisid, donc nous avons déjà les expressions régulières nécessaires. Toutes les données ont été stocké dans un dictionnaire/hash temporaire, pour être par la suite stockés dans la base de données.

3.3 Indexation

Le processus d'indexation est une étape cruciale qui doit être exécutée dans un ordre précis. Il s'agit de nettoyer le body (la partie contenant le texte) du document. On va appliquer le même protocole à chaque mot de chaque body traité. Par la suite nous avons aussi décidé d'indexer les titres, en effet nous avons considéré qu'ils contenaient beaucoup d'informations sur le contenu du document. Les étapes sont les suivantes :

1. Mettre en minuscules les mots
2. Retirer la ponctuation
3. Lemmatisation
4. "Compter" le mot

Les deux premières étapes sont assez évidentes, il suffit de deux expressions régulières.

La lemmatisation est un processus qui a été mentionné en cours mais qui n'a pas été développé par les autres. En effet pendant les TPs nous avons été encouragé à utiliser une troncature à 7 caractères. La lemmatisation utilisé provient du groupe *Snowball*, un projet mondialement reconnu et qui indique des bonnes conventions à respecter. La lemmatisation sous Perl se fait grâce au package *Lingua : :Stem*.

Le processus de comptage des mots est simple. C'est lui qui permet de construire l'index inverse et l'index direct. L'index inverse les mots dans chaque document, l'index inverse fait l'opposé et indique quels document contiennent chaque mot. Lorsque l'on lit un mot dans un document on obtient deux information :

- La mot est dans le document
- Le document contient le mot

Ces affirmations paraissent évidentes mais elles sont au coeur de la structure de notre base de données. En effet, comme nous l'expliquons dans la partie suivante, nous avons eu la possibilité de stocker directement l'index inverse et l'index direct.

Notation : Nous utiliserons désormais le terme <i>lemme</i> au lieu de <i>mot</i> .
--

3.4 Stockage

C'est dans cette partie que nous avons dévié de la trajectoire des autres groupes. Nous avons décidé de ne pas utiliser une base de données relationnelles mais plutôt une base de données structurée, entre autre *MongoDB*. Une description globale de MongoDB est incluse dans l'annexe. On peut néanmoins préciser ses avantages par rapport à ce projet. Une base de données MongoDB est composé de relations qui sont définies par des attributs qui ne sont *pas forcément atomiques*. Cette propriété nous a paru avoir beaucoup de potentiel pour ce projet. En effet on a pu simplement indiquer que l'attribut "sources" d'un document correspond à une liste. Pour comparer, il faudrait créer deux relations en plus pour une base de données relationnelles. Ces propriétés naturelles nous ont fait gagner beaucoup de temps. De plus, puisque il n'y a plus besoin de faire de jointure, la charge sur le serveur est allégé. Le fait de stocker les indexs dans une base de données permet de bénéficier des propriétés des indexs et donc d'accélérer les processus I/O, contrairement aux autres groupes qui utilisent un système de gestion de fichiers non optimisé.

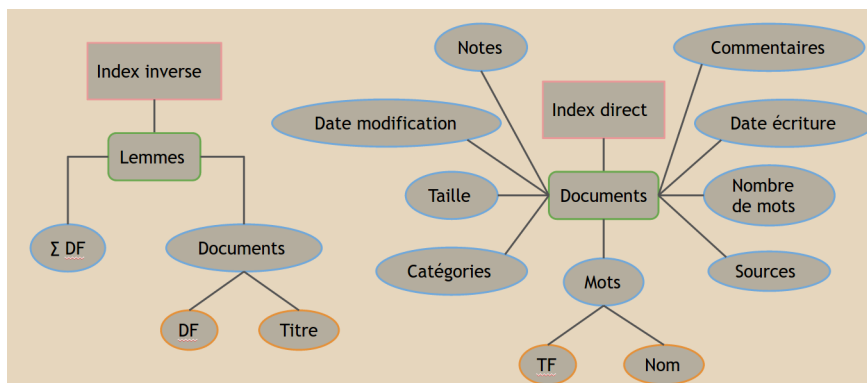


FIGURE 2 – Schéma de la base de données

Comme on le voit sur le schéma, la base de données est limpide et intuitive. Toutes les étapes que l'on a réalisé pendant les TPs ont été sauvegardés dans ces structures. En effet nous avons pu conserver les structures d'index inverse et d'index direct présentés en cours. De plus nous avons enrichi l'index direct pour qu'il puisse contenir les métadonnées propre à chaque document.

3.5 Appariement

L'objectif est de retourner les documents qui correspondent le mieux à une requête utilisateur. Pour cela nous avons appliqué le cours de Recherche d'Informations donné par Mme. Lechani. Grâce à l'index inverse stocké dans la base de données, on peut facilement savoir quels documents contiennent un terme donné. Il suffit alors de le faire pour chaque mot de la requête. Bien sûr les mots subissent le même protocole de nettoyage que lors de l'indexation des données. L'algorithme est alors très simple, on boucle à travers tous les mots de la requête, puis on boucle à travers tous les documents qui contiennent le lemme, auquel on assigne un score RSV. Bien sûr si le document possède déjà un score on l'incrémente. Pour les scores il y'a différents modèles pouvant être utilisés, les algorithmes sont décrits en Perl dans l'annexe :

- Term Frequency Inverse Document Frequency (TFIDF)
- TFIDF Variété (algorithme imaginé par le groupe)
- Okapi BM25
- Fréquentielle
- Booléen restrictif

Le TFIDF est le produit entre la fréquence d'un mot et sa rareté. Il peut exploser si un mot rare est présent beaucoup de fois dans un document, ce qui n'est pas forcément voulu. Par exemple si on cherche "*Castres rugby championnat*" et qu'un article contient 20 fois "*Castres*" et un autre 3 fois "*Castres*", 3 fois "*rugby*" et 2 fois "*championnat*" le TFIDF sera plus haut pour le premier document, alors que la pertinence utilisateur est plus forte pour le second article. C'est en étudiant ces particuliers que nous avons imaginé le facteur *Variété*. L'idée est la suivante : une fois que tous les TFIDF ont été assignés, on parcourt chaque document et on compte combien de mots différents de la requête le document contient, on multiplie alors le TFIDF du document par ce score (la variété).

L'algorithme BM25 (aussi appelé Okapi par rapport au lieu de sa première implémentation) est un algorithme reconnu et très utilisé. Malheureusement nous ne le connaissons pas très bien et nous ne sommes pas étalés dessus. Cependant il est assez simple à coder. On a pu remarquer que les documents renvoyés étaient très pertinents. Il réduit le problème cité du TFIDF mieux qu'en prenant compte de la variété.

Le score fréquentiel est un score très basique. Tout simplement on compte combien de fois chaque mot de la requête apparaît dans un document. Si le document est long il a plus de chance d'être en haut de la liste des documents retournés. De la même façon l'algorithme *Booléen Restrictif* n'est pas très compliqué, il renvoie les documents (sans score de pertinence) qui contiennent tous les mots de la requête.

Par manque de temps nous n'avons pas implémenté les modèles probabilistes et cosinus.

4 Interface en ligne

4.1 Approche et objectifs

Au risque de se répéter, nous voulons rappeler que nous avons décidé de nous écarter des autres groupes pour réaliser un projet plus personnel. Grâce à du travail personnel nous avons réussi à boucler la partie des traitements et du scoring rapidement, même avant que les séances dédiées au projet débutent. Nous avons profité de cette avance pour aussi réaliser notre propre interface en ligne. Tout au long du projet nous ne voulions pas utiliser la taille de notre groupe comme argument pour expliquer les défauts de notre projet. Au contraire nous avons voulu faire le moins d'impasses possibles. Nous sommes donc partis sur une interface minimaliste mais néanmoins visuellement satisfaisante. Aucun des membres du groupes n'avait d'expérience en programmation web dynamique mais nous avons vite compris les notions clés. Nous avons fonctionné de la manière suivante : trois membres du groupe ont conçu les pages HTML statiques, puis dès qu'un membre finissait d'implémenter une fonctionnalité le dernier membre l'intégrait au serveur. Par exemple un membre a travaillé sur les boîtes déroulantes de la page des requêtes statiquement (avec un bloc notes), puis l'a envoyé au membre chargé de les faire fonctionner sur le serveur. Plus précisément, chaque page web du site est générée avec un script Perl. On peut par exemple traiter une requête utilisateur dans le même script que l'affichage HTML. Dans l'affichage HTML, qui n'est rien d'autre qu'une séquence de `print`, on peut interpoler des variables générées par Perl. Un exemple concret est disponible dans l'annexe.

4.2 Solutions

Pour minimiser le temps de codage et se concentrer sur la finition du site web nous avons utilisé le bootstrap fourni par Twitter. Un bootstrap est un ensemble de classes graphiques (notamment du CSS) qui permet de ne pas réinventer la roue. Celui proposé par Twitter est le plus utilisé au monde. Sa documentation est très fournie et de nombreux tutoriels existe sur le web. Les bootstraps sont un exemple parfaits d'outils opensource que l'on se doit de connaître, en effet nous aurions perdu un temps trop important à coder le CSS et le jQuery que l'on a utilisé. Évidemment nous l'avons cuisiné à notre sauce et nous en avons extirpé beaucoup de connaissances. Comme les autres groupes nous avons utilisé un stack serveur, entre autres LAMPP (version Linux de XAMPP). Une fois que l'on a compris comment fonctionne un stack serveur les choses vont très vite, nous détaillons très brièvement les spécificités de LAMPP dans l'annexe. Nous sommes très heureux d'avoir pu travailler avec cet outil, nous avons beaucoup appris.

4.3 Architecture du site

Le site est composé de 5 onglets :

- Requête par terme(s)
- Requête avancée
- Statistiques
- Paramètres
- Informations

Évidemment les deux derniers onglets sont moins importants que les trois autres donc on les décrira très brièvement. Ci-dessous on peut voir la page d'accueil du site. Pour jouer sur l'élégance nous avons décidé de jouer avec les couleurs bleues et vertes.



FIGURE 3 – Page d'accueil du site web

4.4 Requête par terme(s)

Comme on peut le voir sur la figure précédente, une barre de recherche est conçue pour que l'utilisateur tape sa requête. Lorsque l'utilisateur appuie sur le bouton "Rechercher" le coeur du moteur de recherche s'active. Techniquement le bouton envoie une requête au serveur qui appelle un script Perl (`requete.pl`). Le script prend en argument la requête de l'utilisateur. Il s'agit d'appliquer le protocole décrit dans la partie 3. On peut facilement récupérer les documents contenant chaque terme de la requête (l'opération est décrite dans l'annexe). Une fois que l'on a fini le traitement on obtient un hashage/dictionnaire avec pour clé les noms des documents et comme valeurs leur RSV respectif. Après les avoir triés selon leur RSV en ordre décroissant on peut boucler à travers les noms et les afficher dans le navigateur en imprimant des balises HTML. Le résultat est visible sur la figure suivante.



FIGURE 4 – Exemple de requête par termes

Nous avons décidé de générer une liste déroulante pour chaque document retourné. Quand l'utilisateur clique sur un document la liste s'ouvre pour révéler le body du document. Nous avons aussi ajouté la possibilité de noter et commenter un article au travers de fenêtres modales. La fenêtre modale pour commenter affiche les anciens commentaires.



FIGURE 5 – Fenêtre modale pour commenter

Que ça soit un commentaire ou une note l'information est stockée dans la base de données, exactement comme les catégories et les sources de l'article. Pour consulter ces informations nous avons ajouté un bouton qui ouvre une fenêtre modale avec toutes les métadonnées, le nombre de notes et la note moyenne. Une fois que l'on a compris comment le faire pour une informations il est très simple de le faire pour toutes. Nous n'avons pas été exhaustif dans l'affichage mais d'autres informations pourrait très facilement s'intégrer à l'interface. Il est à noter que l'affichage n'est pas dynamique, ce n'est pas le clic des boutons qui commande les requêtes à la base de données, au contraire celles-ci sont faites au chargement de la page. Ce détail peut paraître anodin mais changer cette façon de faire pourrait dramatiquement réduire le temps de chargement de la page, bien que celui-ci soit déjà assez court.



FIGURE 6 – Fenêtre modale d'informations

4.5 Requête avancée

Dans cette partie l'objectif est de permettre de faire des requêtes structurées, entre autres grâce aux métadonnées récupérées des documents. Pour les articles de *Wikinews* il y a à disposition la date d'écriture et les catégories. Cette fonctionnalité n'a pas été traitée théoriquement et nous a demandé

un peu de travail de réflexion. Grâce au codage de la requête par terme(s) nous avons seulement à changer la requête transmise à MongoDB. En effet, alors que la requête par termes s'adresse à l'index inverse, celle-ci utilise les attributs de l'index direct. L'utilisateur peut donc choisir un jour, un mois, une année et une catégorie. Si un des choix n'est pas rentré on autorise n'importe quelle valeur. Concrètement on envoie une requête portant sur les choix des utilisateurs. Par exemple `categorieDemandée in categoriesDocument`. Nous avons décidé de ne pas ajouter une barre de recherche par termes. Ce n'est pas une question de difficulté d'implémentation, au contraire on l'a fait pour la partie précédente. Tout simplement nous nous sommes dit que la requête structurée peut être assez précise d'elle même et doit se différencier de la requête par termes. Le hic de cette décision est que les documents de comportent pas d'ordre de pertinence système. Néanmoins ce n'est pas la fonctionnalité première de notre moteur de recherche, de plus la requête par termes peut retourner les même documents moyennant une requête appropriée.



FIGURE 7 – Exemple de requête avancée

4.6 Statistiques

Nous avons décidé de ne pas appliquer de statistiques aux documents individuels mais plutôt à l'ensemble des documents, pour plusieurs raisons. Tout d'abord les documents ne sont pas très complexes et il n'y a pas vraiment de raisons de les résumer par des statistiques. De plus, ajouter des fonctionnalités au moteur de recherche alourdit la charge du serveur, or notre but est avant tout de préserver la vélocité de ce dernier. De la même façon que du HTML est généré avec Perl, on peut utiliser du JavaScript. Il se trouve qu'aucun n'avait d'expérience en JavaScript, nous n'avons donc été très exhaustif sur les graphes disponibles. Notre démarche n'a pas été très élégante, le problème étant qu'il n'y a aucun moyen propre de traduire des listes Perl en listes JavaScript. Notre solution a été de concaténer les listes Perl sous forme de chaînes de caractères lues par JavaScript. Bien que cela marche, nous n'avons pas accès aux erreurs JavaScript donc le temps de développement explose.

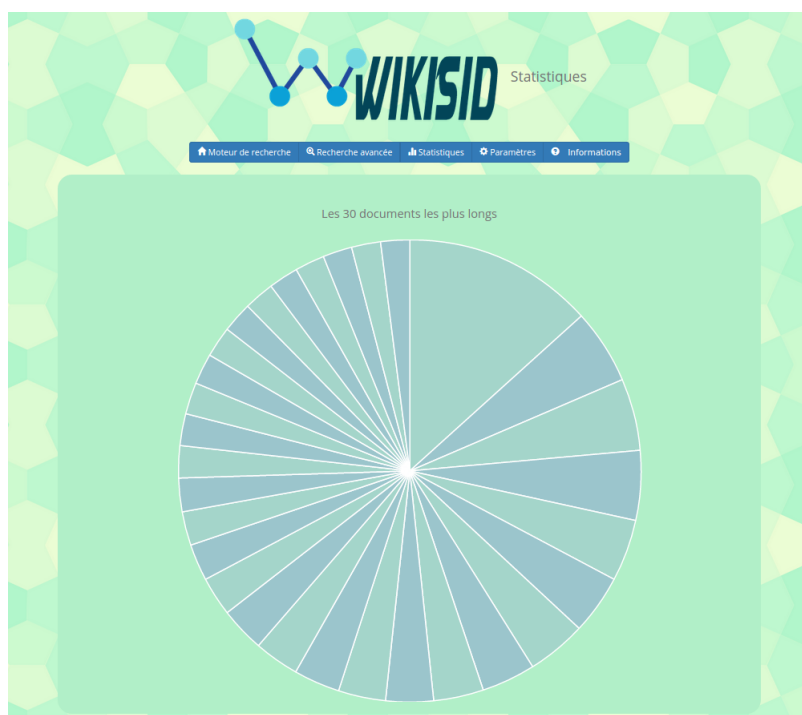


FIGURE 8 – Diagramme circulaire des documents les plus verbeux

Les diagrammes sont interactifs, on peut les survoler et récupérer des informations sur le contenu. Encore une fois nous n'avons pas été très exhaustif et c'est une piste d'améliorations (que l'on liste plus tard dans le rapport).

FIGURE 9 – Histogramme des lemmes les plus utilisés



FIGURE 10 – Nuage de lemmes

4.7 Paramètres

Ce qu'on a fait jusqu'à présent est une méthode archaïque au jour d'aujourd'hui. Des méthodes existent pour générer des pages dynamiques de façons plus élégantes. Un des soucis de notre façon de faire est de déterminer des paramètres. En effet le site a besoin d'un onglet pour déterminer les paramètres des requêtes. Pour adresser cela nous avons décidé de créer un fichier JSON qui contient tous les paramètres du site web. L'utilisateur peut choisir divers options (méthode de scoring, nombres de documents analysés dans l'onglet "Statistiques"...). Lorsque l'utilisateur appuie sur le bouton "Valider" le serveur écrit les paramètres dans le fichier JSON. Lorsque une recherche est lancée ou que la l'onglet "Statistiques" est ouvert le script Perl correspondant commence par lire le fichier JSON et modifie son exécution.

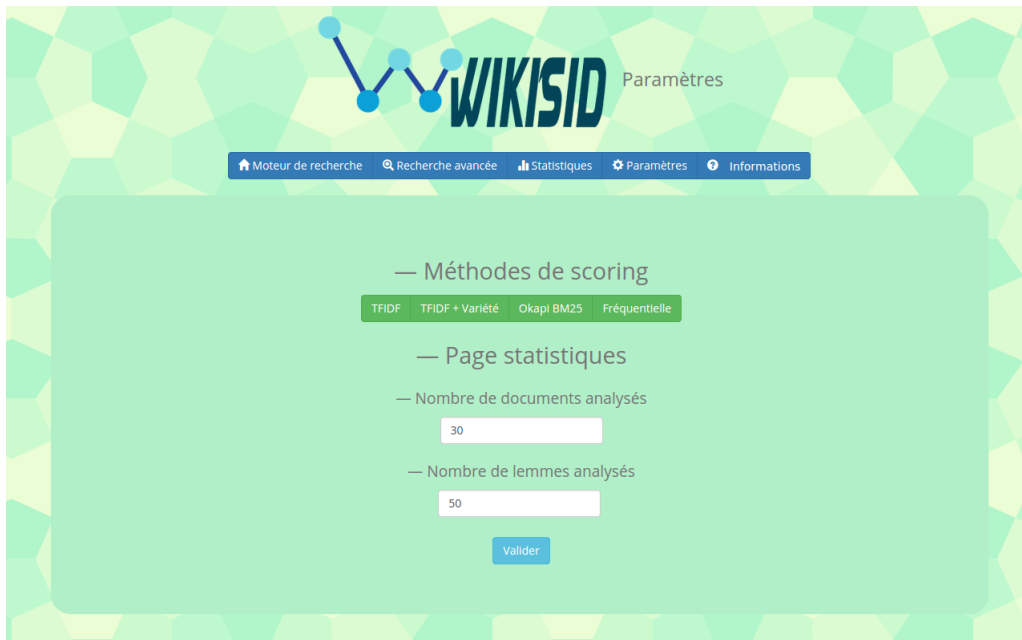


FIGURE 11 – Disposition de l'onglet "Paramètres"

4.8 Informations

Cet onglet contient un bref descriptif du site web pour un visiteur lambda. Il contient aussi un lien vers le code source du projet.

5 Performance du SRI

Nous n'avons pas vraiment effectué de contrôle qualité. Nous avons souhaité garder les contrôles et les revues au strict minimum et plutôt discuter des objectifs en temps réel, sans réunion prévue et planifiée. Comme nous n'avons pas beaucoup de main d'oeuvre et donc pas beaucoup de temps nous nous sommes concentré sur une bonne préparation de chaque phase du projet. Par chance chaque phase s'est réalisée sans souci. Notre projet est le fruit d'une bonne préparation et de travail efficace. En fixant la barre assez haut et en définissant bien le cadre de chaque phase nous n'avons pas fait d'erreur. Cependant nous avons détecté deux mesures de performance de notre SRI.

5.1 Pertinence / Rappel

La première mesure de performance est la précision et le rappel du système. Malheureusement nous ne sommes pas attardés sur ce point, en effet nous avons développé 5 méthodes de scoring et tous les comparer aurait pris du temps mais aurait été souhaitable.

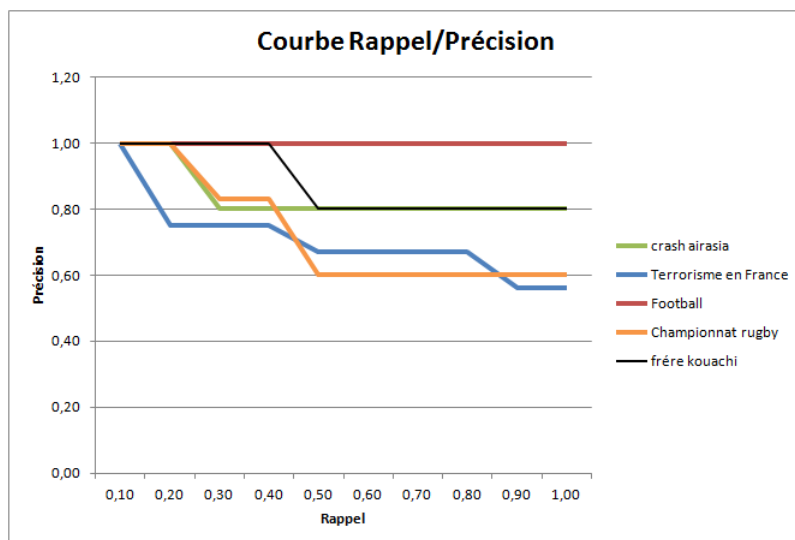


FIGURE 12 – Exemple de graphe des courbes de Rappel/Précision pour la méthode TFIDF

5.2 Temps de réponse

Le temps de réponse de chaque requête est stocké de la façon suivante : un dossier contient des fichiers textes qui stocke les temps de réponse pour chaque méthode de scoring et pour chaque longueur de document. Nous avons jugé que c'était une façon pertinente de conduire l'analyse du temps de réponse. En effet le nombre de termes de la requête (nettoyée) est directement lié au temps de réponse : plus de termes = plus de document à traiter. Les algorithmes de scoring sont à complexité différente, il faut en tenir compte. Tous les temps de réponse sont disponibles dans le dossier *performance*. Nous avons écrit un script Python pour pouvoir les analyser. Par exemple une analyse interactive du *TFIDF Variété* est disponible à ce lien : <https://plot.ly/~MaxHalford/26/tfidf-variete/>. Nous n'avons pris le temps d'analyser en profondeur chaque algorithme mais le script (très simple et modulable) et les données sont en place.

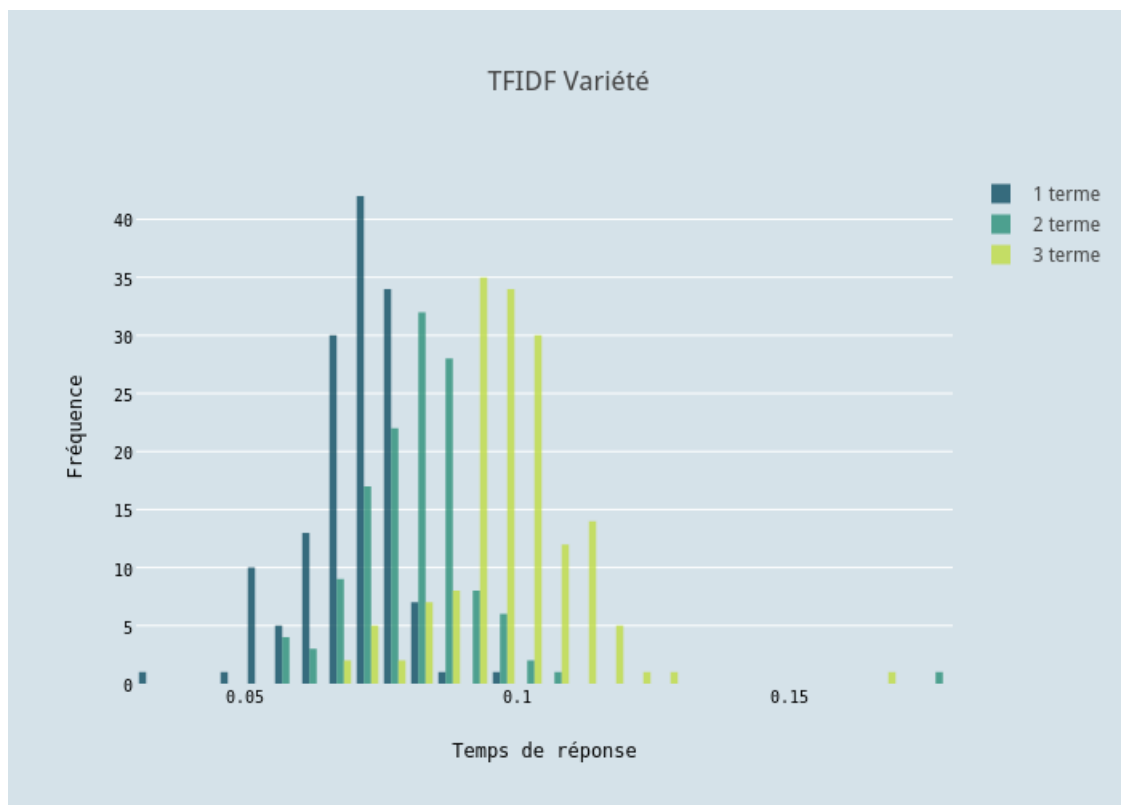


FIGURE 13 – Exemple d'analyse des temps de réponse

6 Bilan

6.1 Le produit

Nous pensons avoir satisfait le cahier des charges. Le processus de nettoyage et d'indexation marche parfaitement. Le lien entre les données récupérées et MongoDB a été fructueuse, en effet le schéma de la base de données a naturellement découlé de la façon dont on a extrait les données et permet un accès direct aux données nécessaires. L'interface web est ergonomique et affiche les informations nécessaires. Nous pensons ne pas avoir fait d'impasse et que l'utilisateur hypothétique aura une agréable expérience agréable.

6.2 Propositions d'améliorations

- Se rendre indépendants des sites webs étudiés se concentrer sur l'indexation des bodys.
- Packager le code pour qu'il soit plus clair et plus modulable.
- Coder les notes, les commentaires, les paramètres et l'affichage des documents en AJAX.
- Mettre en place des requêtes en batch pour minimiser les aller/retours à la base de donnée.
- Paginer les documents retournés.
- Tuner les deux hyperparamètres de la méthode Okapi BM25 (algorithme génétique?).
- Mettre en place une procédure automatique interactive pour que l'utilisateur compare les méthodes de scoring.
- Stocker les bodys dans la base de donnée sous forme de *blobs*.
- Réduire la quantité d'informations calculée à l'affichage pour les rendre activables par des fonctions JavaScript.
- Réduire la taille des identifiants des documents (recoder en hexadécimal par exemple).
- Pousser l'analyse de pertinences des documents retournés.

6.3 L'équipe

6.3.1 Rémy Simon

Pour ma part ce projet aura été un vrai challenge, il m'aura mis face à de vraies difficultés notamment lorsque nous avons du manipuler les différents langages de programmations où je n'étais pas le plus à l'aise. Cependant je suis plutôt fier du résultat et tiens à remercier les membres de mon groupe sans qui je n'y serais probablement pas arrivé, et notamment Max qui a su nous diriger tout au long de la réalisation du projet.

6.3.2 Max Halford

Ce projet a été riche en savoir pour ma part : pages HTML dynamiques, jQuery, CSS, traitement de fichiers... Je suis aussi très content d'avoir pu me perfectionner avec MongoDB, c'est mon deuxième gros projet avec et j'ai corrigé beaucoup d'erreurs par au premier. Bien qu'elle soit fascinante, le côté recherche d'informations m'a un peu moins plus, j'ai adoré traité les fichiers et indexé les données, mais je trouve que les métadonnées n'étaient pas nécessaires. En effet je pense qu'on ne s'est pas assez penché sur les méthodes de scorings et sur la pertinence des document retournés. Pour ma part j'aurais préféré construire une application qui analyse les bodys de n'importe quelle page web et implémente une méthode de scoring robuste. Cependant il y'a eu bien plus de positif que de négatif dans ce projet. J'aimerais aussi ajouter que je suis très content de notre équipe, je ne cache pas le fait que je suis fier de notre travail et du fait que nous ayons réussi à quatre ce que d'autres, sans connotation négative, n'ont pas entièrement fini.

6.3.3 Kafil El Khadir

Ce projet a été pour moi une vraie expérience à travers laquelle j'ai pu voir ce qu'un travail en groupe dans une entreprise peut être, en général je suis plus que satisfait du rendement de notre groupe ,du travail réalisé et ainsi que le résultat final . La coordination des tâches qu'on a pu effectuer pour la réalisation de ce projet était sans aucun doute le facteur clé de cette réussite, ce dernier n'aurait pu se faire sans l'effort indispensable de notre chef de groupe Max Halford qui a su gérer l'ensemble des tâches ainsi que leurs affectations tout en gardant une bonne ambiance au sein du groupe. Je rejoint Max sur sa dé-plaisance au niveau du domaine de la recherche d'information qui n'est pas aussi passionnant que d'autre d'enseignement qu'on a eu dans la l3 SID, néanmoins grâce à son idée pour changer de langage de gestion de Base de données j'ai pu découvrir MongoDB qui a été pour nous un atout essentielle dans la réussite de ce projet. Pour conclure je tiens à remercier aussi Remy Simon et Salima Azzou pour leurs travail et souligner le grand intérêt qu'ils ont porté à ce projet.

6.3.4 Salima Azzou

Ce projet m'a permis de découvrir différents langages de programmation, de plus notre chef de projet (Max Halford) a pris l'initiative d'utiliser et de nous initier à MongoDB, un SGBD qui nous a permis de nous faciliter l'étape de stockage de métadonnées. Je suis très fière et satisfaite du résultat final, je voudrais également mentionner la cohésion du groupe et les remercier pour leurs investissements au projet.

Références

[SITE] Documentation MongoDB.

[SITE] Driver MongoDB pour Perl.

[REF] Karbasi. *Pondération des termes en Recherche d'Information*, 2007.

[SITE] Bootstrap de Twitter.

[REF] Chavelli. *Prenez en main Bootstrap*, 2015.

Ces références sont cliquables et redirigent vers des sites web.

Le projet dans sa totalité est disponible à l'URL suivante :

`https://github.com/MaxHalford/Wikisid`

Il suffit de mettre le dossier master dans le dossier *htdocs* de XAMPP/LAMPP. Pour que tout fonctionne correctement il faut aussi suivre les étapes données dans l'annexe, notamment installer les modules.

7 Annexes

7.1 MongoDB

MongoDB est un système de gestion de bases de données faisant partie de la mouvance noSQL. C'est un logiciel très populaire, en effet c'est le quatrième plus utilisé au monde et le premier en termes de croissance. Son approche semi-structurée des données permet d'adresser des cas d'utilisations avec une approche élégante et intuitive. Contrairement aux bases de données relationnelles MongoDB n'assure pas l'indépendance des données à l'utilisation qu'on en fait, au contraire il nécessite de structurer les données selon les besoins qu'on en a. Il n'est donc pas question de normaliser les relations, mais plutôt de concevoir la base de données en fonction de l'utilisation qu'on en aura. Cette approche demande de l'imagination et peut être la source d'erreur. Cependant cette façon de faire a beaucoup d'avantages si l'on connaît les données stockées. De plus MongoDB, de par sa nature semi-structuré, facilite l'implémentation des relations.

La principale différence avec les bases de données relationnelles est que les attributs des relations ne sont pas forcément atomiques. MongoDB stocke les données de façon arborescente, exactement comme un fichier JSON. D'ailleurs les opérations sur la base de données sont aussi similaires à des fichiers JSON. Un exemple, dans le cas de Wikinews, est qu'un article peut contenir plusieurs sources et plusieurs catégories. Avec une approche relationnel on crée une relation pour chaque attribut et une autre table pour relier les pages aux attributs, ce qui fait 5 tables. Avec MongoDB on peut tout simplement dire qu'un document possède plusieurs sources et plusieurs catégories en les mettant dans une liste. Cette liste est similaire à une liste dans n'importe quel langage de programmation. Avec cette approche il suffit d'une collection (équivalent des tables en SQL).

La puissance des bases de données relationnelles provient des liens entre les relations, c'est avec les jointures que les données prennent un sens. MongoDB n'a aucune notion de clé étrangère. Si l'on veut associer une branche d'une collection à la branche d'une autre, il faut passer par un langage de programmation et écrire la jointure manuellement. Cependant l'objectif est de concevoir une base de données qui évite de nécessiter des liens entre les collections. L'avantage de contenir toutes les données dans une seule collection est la rapidité des requêtes, elles sont extrêmement rapides grâce à l'absence de jointures. Le revers de la pièce est qu'il peut exister une redondance non négligeable si l'arborescence des collections est trop profonde. Pour résumer, MongoDB peut aller plus vite que SQL mais nécessite plus de place sur le disque.

Dans notre cas les données étaient parfaitement adaptés pour MongoDB, c'est même un cas d'école. L'index inverse et l'index direct étant des hashages, les stocker dans une collection ne nécessite aucune transformations puisque c'est le format que MongoDB utilise.

Pour utiliser MongoDB il suffit de l'installer. Pour une utilisation plus agréable il vaut mieux utiliser un driver. Un driver permet de communiquer entre un langage de programmation et une instance MongoDB. Cependant les drivers ne sont pas édités par la compagnie qui maintient MongoDB donc il est assujéti à un manque de fonctionnalités. On peut citer le driver *py-mongo* pour le langage Python qui est le driver le plus complet actuellement. Malheureusement nous nous sommes rendus compte que le driver officiel pour Perl manque cruellement de fonctionnalités, notamment pour mettre à jour des données.

On aimerait souligner que le temps de développement avec MongoDB est extrêmement court. En effet il n'y a pas besoin de préciser les types des attributs ni les contraintes relatives aux relations, en effet celles-ci doivent être implicites et découle du traitement des données fait auparavant. Encore une fois MongoDB fait confiance à l'utilisateur, l'avantage est que les collections sous MongoDB ont moins d'options à gérer, ce qui minimise les temps de calcul. Nous pensons que pour un projet de petite envergure et où les données ne sont pas complexes, MongoDB l'emporte largement sur SQL en termes de temps de développement et de facilité d'implémentation.

7.2 LAMPP

LAMPP est la version Linux de XAMPP. C'est un *solution stack*¹, en d'autre termes un programme qui contient tout ce qui est nécessaire pour héberger des pages web. Un solution stack est indispensable pour héberger des scripts qui génèrent du HTML, en effet un navigateur ne peut, par défaut, qu'afficher des pages webs statiques. Pour que le programme fonctionne il faut ajouter des modules. Sur Linux cela se fait avec les commandes suivantes :

- `/opt/lampp/bin/perl -MCPAN -e 'install MongoDB'`
- `/opt/lampp/bin/perl -MCPAN -e 'install Lingua : :Stem'`
- `/opt/lampp/bin/perl -MCPAN -e 'install Digest : :MD5'`

Normalement les autres modules utilisés sont disponibles par défaut avec LAMPP, si jamais un module n'est pas incluse il suffit de relancer les commandes précédentes avec le nom du module.

1. https://www.wikiwand.com/en/Solution_stack