
# Rapport de projet

---

Quentin Couavoux
Léo Menaldo
Alexandre Colliard
Justine Gueulet

*EPITA 2024* *Projet s4*

## Table des matières

# 1 Plan de soutenance

## Introduction

## Démonstration

- User Input Processing
- Crawler
- Scraper
- Interface

## Conclusion

- Éléments terminés
- Ressenti global

# 2 Prélude

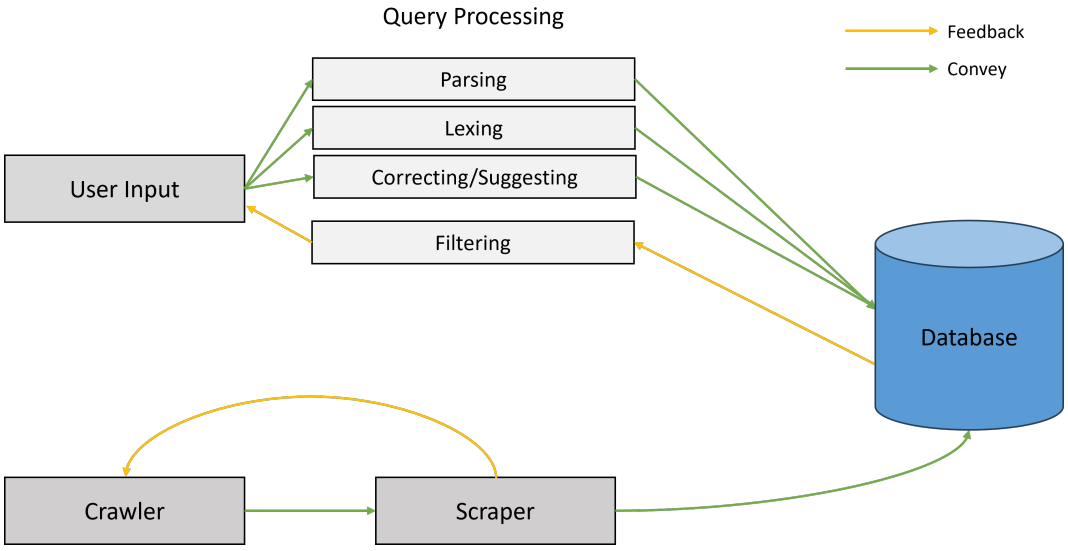
## 2.1 Le projet

En 1990, Alan Emtage crée le premier moteur de recherche afin de faciliter la navigation sur internet pour les utilisateurs. Depuis, plusieurs moteurs de recherche étant fondés sur le travail d'Alan Emtage ont été conçus pour aider les utilisateurs à faire leurs recherches aisément tout en répertoriant les sites web sur Internet. Or, malgré les avancements technologiques de ces moteurs de recherche, seulement 85% des utilisateurs obtiennent des informations précises qui répondent à leur demande. Notre groupe a donc décidé de s'intéresser à ce problème.

Pour notre projet, nous introduisons un moteur de recherche stable et rapide, tout en répondant de manière précise aux requêtes des utilisateurs.
Un moteur de recherche peut être séparé en deux parties principales : celle du "User Input" et celle du "Crawler". Ces parties fonctionnent simultanément afin de permettre à l'utilisateur d'effectuer ses recherches tout en agrandissant la base de données.

Le moteur de recherche permet à l'utilisateur de rechercher des informations via des requêtes. Ces requêtes sont ensuite gérées par le "Query Processing" où les mots clés de la requête sont déterminés. Il est possible de proposer des suggestions et/ou des corrections d'orthographes en cas de mots clés mal écrits ou erronés afin de permettre à l'utilisateur d'obtenir de meilleurs résultats. Une fois la requête traitée, les sites Web répondant au mieux à la demande de l'utilisateur sont récupérés de la base de données, triés en fonction de leur pertinence et envoyé à l'utilisateur.

La partie "Crawler" possède deux catégories : le "Crawler" et le "Scraper". Le Crawler est le processus de recherche et de récupération itérative de liens de sites Web à partir d'une liste d'URL de départ. Le Scraper, quant à lui, récupère les liens visités pour extraire et d'envoyer au Crawler les URLs référencées dans ces derniers pour continuer l'exploration Web. Le Scraper se charge également de traiter les documents Web et d'en extraire des informations afin d'y envoyer à la base de données.



## 2.2 Concurrents

### 2.2.1 Google Search

Google Search est un moteur de recherche possédant trois principales unités fonctionnelles : le Googlebot, le Query Processor et l'indexer. Le Googlebot, composé de nombreux ordinateurs qui envoient fréquemment des requêtes à des milliers de sites Web à la fois, parcourt ces sites et transfère les informations collectées vers la base de données. Lors du classement des sites dans l'indexer, Google prend en compte un certain nombre de facteurs, notamment la pertinence et l'expérience utilisateur. Google accorde également une grande importance aux liens entrants et à la pertinence des sites Web liés, ainsi qu'à l'optimisation des pages.

### 2.2.2 Yahoo !

Yahoo ! est un moteur de recherche qui utilise une combinaison d'éditeurs humains et d'algorithmes pour classer les sites. Tout comme le Googlebot, le Yahoo ! Slurp parcourt des sites Web tout en les sauvegardant dans l'indexer. De plus, les éditeurs humains de Yahoo ! examinent et catégorisent manuellement les sites Web, ce qui contribue à garantir l'exactitude et la pertinence des résultats. Le moteur de recherche prend également en compte des facteurs tels que la pertinence des informations du site par rapport à la requête de recherche de l'utilisateur, la qualité du contenu et la qualité des liens entrants du site.

### 2.2.3 Bing

Bing est un moteur de recherche utilisant une approche de classement des sites différente de celle de Google. Le Bingbot est utilisé pour parcourir les sites Web tout en les ajoutant dans la base de données après les avoir classifiés par critères. Or, Bing met davantage l'accent sur l'expérience utilisateur, notamment la qualité du contenu et la facilité de navigation, ainsi que la pertinence des informations par rapport à la requête de l'utilisateur. Bing prend également en compte la qualité des liens entrants du site, ainsi que son autorité globale et sa popularité sur le Web.

### 2.2.4 Baidu

Baidu est un moteur de recherche chinois qui est surtout destiné à la population et au marché chinois. Son fonctionnement est similaire à celui de Google Search, or, les robots d'exploration de Baidu ne peuvent comprendre que le contenu textuel et ont du mal à explorer le contenu rendu sous forme d'image. Cela signifie que ce contenu peut ne pas être indexé ou classé.

Google Yahoo ! Bing Baidu

# 3 Présentations

## 3.1 Quentin Couavoux

Bonjour, je me présente en tant qu'étudiant en informatique, actuellement en deuxième année à l'EPITA. Mon intérêt pour l'informatique remonte à l'âge de 10 ans, une époque où je m'initiais déjà à la programmation sur Minecraft. Cette passion m'a accompagnée tout au long de mon parcours, m'incitant à m'engager davantage dans ce domaine. À l'heure actuelle, je suis impatient à l'idée de débuter le projet de S4, suite logique des projets de S2 et S3 qui se sont déroulés de manière singulière et enrichissante. Chaque étape m'a permis de développer mes compétences et d'explorer divers aspects de l'informatique. Ce qui me motive particulièrement dans le projet actuel, c'est l'aspect algorithmique. La perspective de travailler sur des algorithmes complexes. Ainsi, je suis prêt à relever les défis et à contribuer pleinement au succès de ce projet. Que l'aventure commence.

## 3.2 Léo Menaldo

Joueur de jeux vidéo depuis mon plus jeune âge, j'ai très vite su que j'allais m'orienter vers l'informatique plus tard. Les années ont passé et j'ai intégré l'EPITA. Cela fait maintenant une année et demie que j'y suis et que je m'y investis, les moments les plus mémorables restent les périodes de projets. Que ce soit le jeu vidéo fait durant le deuxième semestre ou le terrible solveur de sudoku du troisième, j'ai su surmonter ces épreuves en restant soudé avec mon groupe. Ce projet s'annonce comme quelque chose de coriace qui va nous défier sûrement bien plus que les précédents projets.

## 3.3 Alexandre Colliard

Passionné d'informatique depuis la seconde, je me suis entraîné dans de nombreux projets ambitieux, seul ou avec des amis. J'ai déjà codé un jeu Android multijoueur, que j'ai pu déployer sur le Play Store, mais il n'a jamais vraiment abouti. Le projet S2 a été pour moi un moyen de découvrir la gestion de projet dans un cadre sérieux et professionnel (ici scolaire). Le projet S4 sera l'occasion de pousser l'aspect algorithmique des applications et d'appliquer les notions d'optimisation et de rapidité. J'espère que notre expérience sur les différents projets de groupe (S2,S3) nous permettra de mieux gérer notre temps pour finir l'application. Marge !

## 3.4 Justine Gueulet

Depuis mon plus jeune âge, le monde de l'informatique m'a toujours intéressé. Mon père travaillant dans l'informatique, je me suis souvent amusée à l'imiter. Cette passion n'a fait que croître lors de la spécialité NSI, surtout l'intelligence artificielle et la robotique. J'ai toujours voulu faire un projet qui se concentre sur les moteurs de recherche avec du Natural Language Porcessing (NLP). Or, il faut tout d'abord avoir des connaissances sur le fonctionnement des moteurs de recherche. C'est pour cela que j'ai vraiment hâte de commencer ce projet avec mes camarades de classe !

# 4 Assignation des tâches

Ci-dessous, les tâches assignées à chacun des membres du groupe lors du projet :

## 4.1 Quentin Couavoux

Pour cette soutenance, , l'interface doit être opérationnelle du point de vu graphique.

## 4.2 Léo Menaldo

Un crawler, outil permettant de parcourir internet, a dû être implémenté.

## 4.3 Alexandre Colliard

Pour cette soutenance, l'interface doit être opérationnelle avec toutes les fonctionnalités que l'on attend d'un moteur de recherche, ainsi que le lien avec notre base de données doit être fait.

## 4.4 Justine Gueulet

Pour cette soutenance, la base de données doit être implémentée et utilisable.

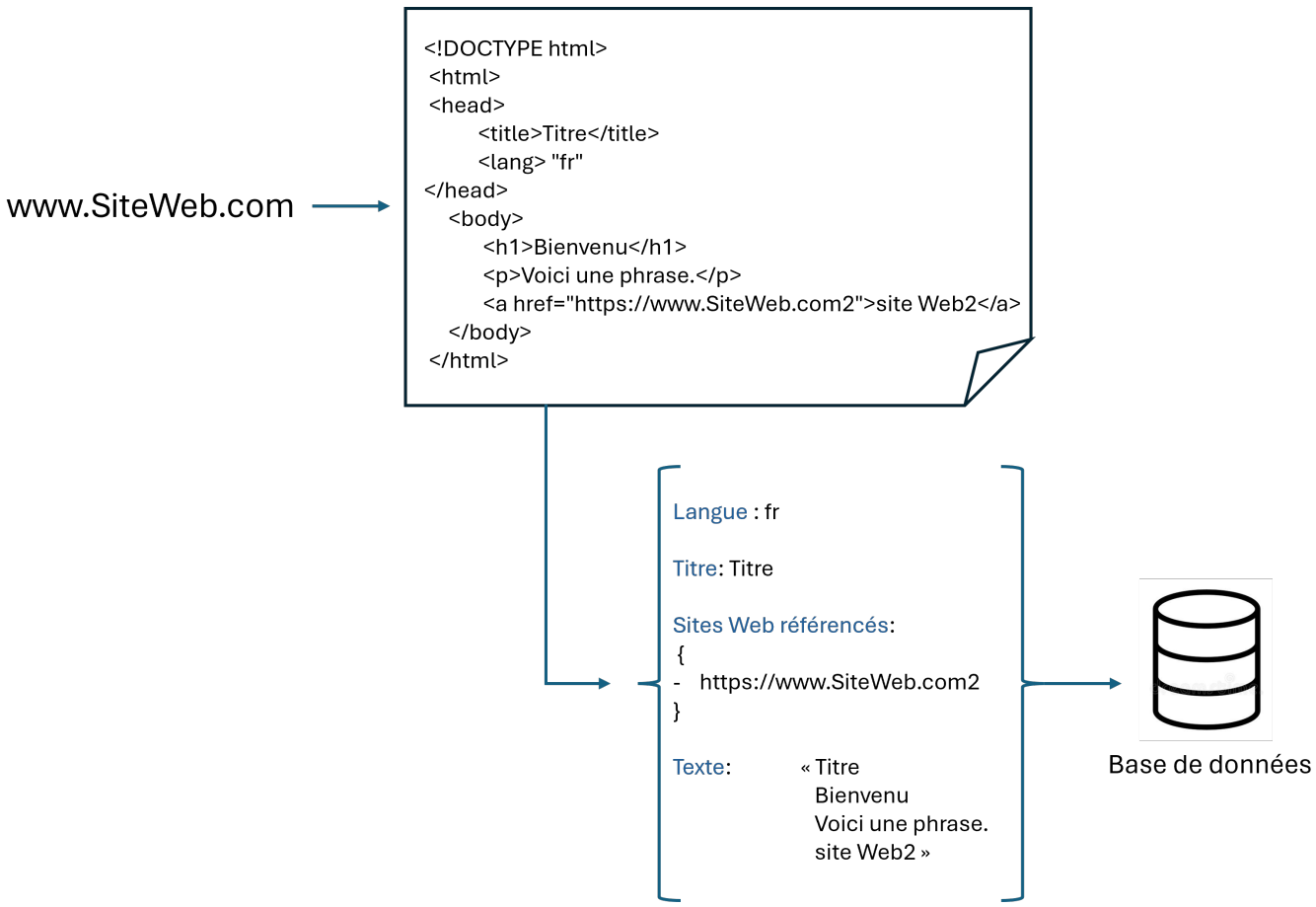
# 5 Scraper - Justine

## 5.1 Scraper

Lorsqu'un utilisateur effectue des recherches sur un navigateur Web, il obtient des résultats correspondant à sa demande. Or, comment est-ce que le navigateur Web reconnaît quels sites Web sont pertinents ? Pour ce faire, il utilise des technologies telles que les Scraper permettant d'obtenir une grande quantité d'informations à partir d'un site Web en très peu de temps.

Après avoir été obtenu grâce au Crawler, le Scraper récupère un site Web et récupère son code source, qui est généralement en HTML5. Il s'occupe par la suite de trier les informations trouvées dans le code HTML comme le titre, la langue, paragraphes, etc parmi les différentes balises du code HTML. Une fois toutes les données cruciales récupérées, il applique des algorithmes tels que le TF-IDF afin de savoir quels mots sont récurrents ou non dans le site Web, ou dans certains cas, de l'intelligence artificielle est utilisées en plus.

Pour finir, le Scraper ajoute ce site Web dans la base de données avec toutes les informations recueillies le décrivant, permettant à l'utilisateur de retrouver ce site Web lors de futures recherches.

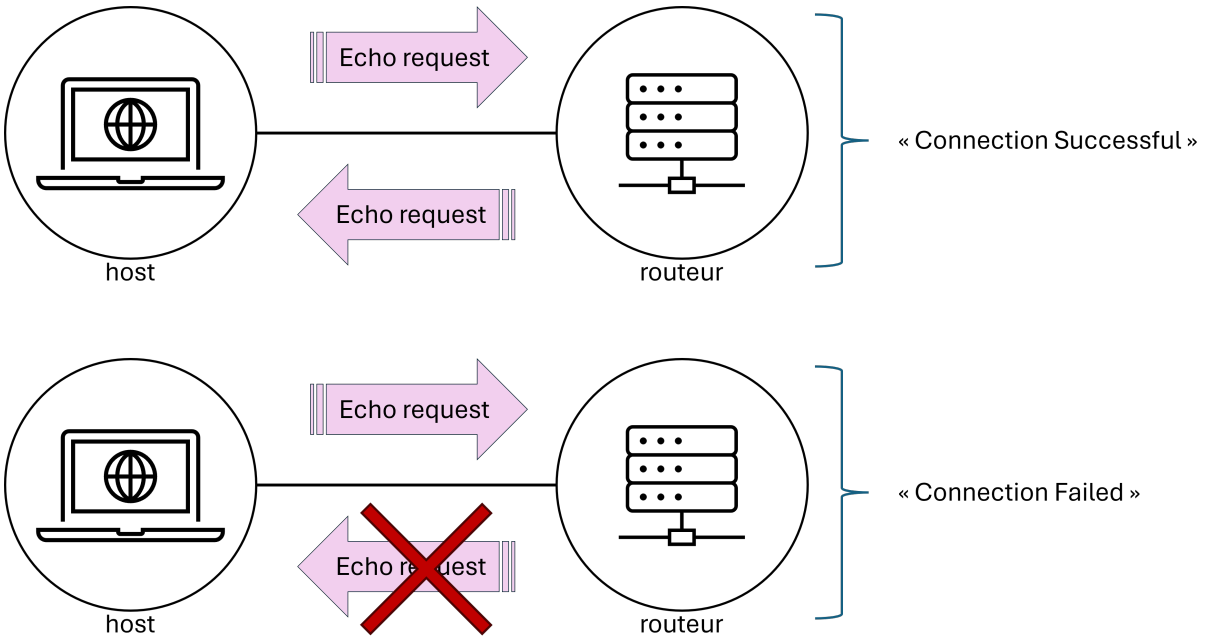


## 5.2 Récupération du code HTML à partir de l'URL

Le Scraper est une partie qui est censée récupérer le code source en HTML d'une URL pour pouvoir ensuite l'analyser et appliquer certains algorithmes sur ce dernier. Or, lors de la première soutenance, la partie du Scraper a rencontré quelques problèmes de librairies Rust et n'arrivait pas à implémenter correctement la fonction permettant la conversion. La librairie que nous avions essayé d'utiliser était la librairie HTTP Hyper qui est rapide et stable. Or c'est aussi une librairie très low-level qui est assez compliquée à prendre en main d'où notre incapacité à faire marcher notre fonction de conversion.
Après plus de recherche, sachant que cela n'était pas une priorité pour notre première soutenance, nous avons trouvé plusieurs librairies HTTP Rust permettant d'effectuer ce que nous voulions faire de manière toute aussi rapide et beaucoup plus simple. La librairie HTTP que nous avons utilisé pour le Scraper est donc la même librairie que nous utilisons pour le Crawler, ce qui nous évitera d'avoir plusieurs librairies servant la même nécessité.

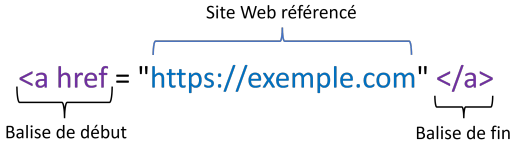
Une fois la fonction implémentée, lorsque nous n'avons pas de connexion internet la fonction crée une boucle infinie ce qui pose problème. Il a donc fallu implémenter une fonction supplémentaire afin de vérifier si nous captons une connexion internet. Si nous sommes connectés à un réseau, la fonction de conversion d'une URL au code HTML se lance sans soucis, sinon nous obtenons une erreur immédiatement nous prévenant de notre manque de connexion internet. Pour cela, nous faisons appel au protocole ICMP (Internet Control Message Protocol), où l'hôte et les routeurs échangent des données avec l'hôte qui renvoie une erreur si les données n'ont pas été reçues intactes.

Cette fonction rajoute une sécurité afin d'éviter que notre moteur de recherche émette des erreurs à cause d'un manque de connexion qui crée une boucle infinie.



## 5.3 Extraction des URLs

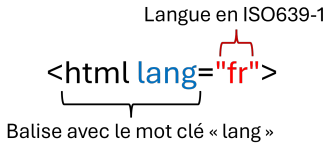
Une pratique commune lors de la création d'un site web est le référencement de sources, généralement des sites web, laissant à un utilisateur la possibilité de découvrir de nouveaux sites. Pour accomplir cela, il existe en langage HTML des balises. Ces dernières sont utilisées pour formater, structurer et hiérarchiser les données tout en permettant au navigateur de savoir comment afficher les informations.
Pour référencer un site web dans un code source HTML, il est nécessaire d'utiliser la balise "<a href>" suivie d'un lien valide vers une autre page web.




Il est également possible d'utiliser ces balises pour faire remonter l'utilisateur sur une partie spécifique de la page web ou l'amener sur une autre section du site web. C'est pour cela que j'ai décidé de chercher les emplacements de ces balises référençant d'autres sites web et de les ajouter à un vecteur au fur et à mesure de mon parcours dans le code source HTML du site web, pour renvoyer un array possédant tous les liens rencontrés dans le code source. Or je me suis vite rendue compte que certaines URLs que j'ajoutais dans mon vecteur ne menaient pas vers d'autres sites. pour parer à ce problème, j'ai décidé d'implémenter pour le moment une fonction vérifiant le format des liens récupérés. Or pour les prochaines soutenances je voudrais faire en sorte de n'avoir que des URLs valides et utilisables afin de pouvoir envoyer directement ces URLs à la partie concernant le crawler. Pour ce faire je vais devoir me renseigner sur les balises de référencement ainsi que sur les différentes version du langage HTML et d'autres langages utilisés tels que le XHTML.

## 5.4 Extraction de la langue

Lorsqu'un utilisateur effectue une recherche, le navigateur va devoir détecter plusieurs critères : Certains mots clés, le PageRank d'un site web, . . .Mais la langue est aussi un de ces critères et elle joue un rôle important dans la qualité des résultats renvoyés à l'utilisateur. Pour renseigner la langue, une balise "<html lang="fr">" est utilisée en début de code.



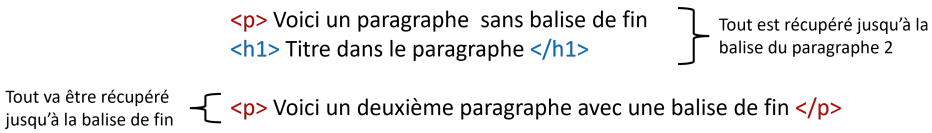

Il faut donc rechercher le mot clé "lang" pour obtenir la langue. Or la langue doit être encodé en ISO639-1, il faut donc effectuer une vérification de la langue. Pour ce faire, il faut regarder si cette dernière fait partie des 217 abréviations de langues.

## 5.5 Extraction des paragraphes

Contrairement aux liens référencés ou que la langue utilisée où il existe une balise particulière pour indiquer au compilateur leur nature, les textes sont plus rudes à gérer. Certains langages de programmation balisés possèdent des conventions qui divergent des autres, ce qui peut rendre la gestion de tous les cas distincts difficile. De manière générale, pour indiquer le début d'un paragraphe, une balise "<p>" est utilisée. Cette dernière signale au compilateur qu'il doit formater d'une certaine façon les données contenu dans cette balise afin d'avoir un rendu concis et clair. Sans cette balise, le navigateur risque de mal mettre en place le texte et peut créer des décalages voire une disparition du texte. Or la balise de paragraphe <p> n'est pas l'unique balise indiquant un texte, il existe également :

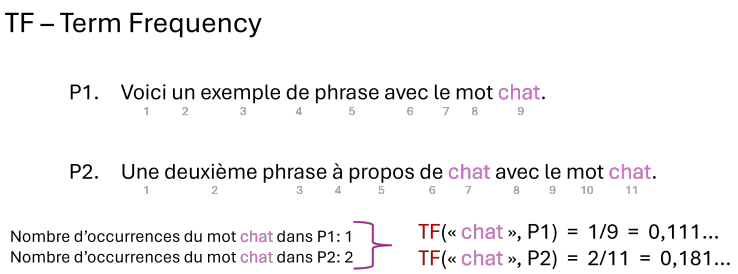
— $< DL >$ indique le début d'une liste de descriptions et peut comporter d'autres balises de mise en page.
— $< H_i >$ avec $i \in [\![1;6]\!]$ représente un titre, le chiffre indiquant sa taille.
— $< TT >$ (obsolète) permet d'afficher un élément à largeur fixe tel qu'un téléscripteur.

Les langages de programmation à formatage possèdent une quantité assez surprenante de balises de mise en page, et il faut que nos algorithmes puissent toutes les détecter ainsi que gérer tous les cas possibles. Il faut donc, pendant l'analyse du code source, prendre en compte chacune de ces balises bien spécifiques puis les retirer du paragraphe afin d'avoir un texte propre.
Or, certaines de ces balises sont considérées comme obsolètes et les sites possédant ces dernières courent le risque d'être supprimés. Malgré ce risque, sachant que ces sites sont encore en ligne, il faut tout de même les analyser et gérer le cas des balises obsolètes. Il existe aussi la possibilité de mettre certaines balises de mise en forme à l'intérieur d'autres balises de formatage. On peut par exemple mettre une balise de titre dans une balise de paragraphe. Mais la plus grosse difficulté rencontrée durant l'extraction de paragraphes a été de prendre en compte les différentes versions de langages de programmation de balisage. En utilisant le XHTML, il est obligatoire de fermer une balise de paragraphe <p> avec < /p>, sinon une erreur de compilation risque d'apparaître. Tandis qu'en HTML5 il n'y a aucune obligation d'utilisation une balise fermante pour indiquer la fin d'un paragraphe. Pendant la compilation du code source d'un site Web, la fin d'un paragraphe est indiquée par le commencement d'un autre paragraphe ou avec d'autres balises. Sachant qu'il pourrait être encombrant de créer plusieurs fonctions gérant ces cas séparément, il est plus pratique et simple pour nous de ne se focaliser que sur une unique fonction, surtout que le XHTML est assez peu utilisé de nos jours. Nous commençons par récupérer les emplacements des balises de début. S'il existe une balise de fin, nous récupérons le texte jusqu'à cette dernière, sinon nous continuons jusqu'au début de la prochaine balise d'entrée.

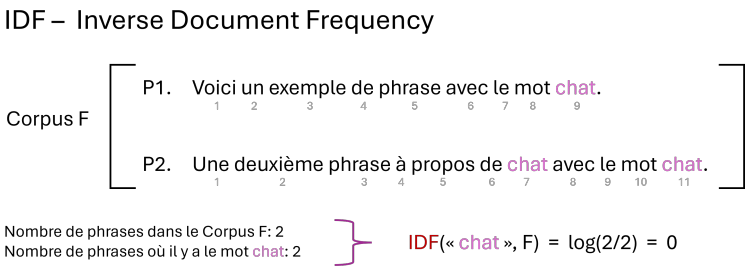


## 5.6 TF-IDF

Le "TF-IDF", aussi connu sous le nom de "Term Frequency * Inverse Document Frequency", est une méthode de SEO (Search Engine Optimization) permettant de trouver les mots les moins utilisés sur une page Web aisni que les mots les plus utilisés. Pour ce faire, il est séparé en deux parties distinctes : le TF (Term Frequency) et le IDF (Inverse Document Frequency).




Le TF permet de calculer la fréquence d'un mot dans un document, donc dans notre cas dans une page Web. Il va compter le nombre d'occurences du mot pour ensuite le diviser par le nombre de mots en tout sur la page Web. Faire cela permet de voir si un mot est important au sein d'une certaine phrase ou non, ainsi que de voir sa pertinence globale sur la page Web. Or cela ne suffit pas pour savoir si ce mot en particulier est un mot clé. C'est pour cela que nous devons utiliser la partie IDF.




La partie de l'IDF s'occupe de regarder la rareté du mot choisi dans un corpus, soit dans notre cas une page Web. Il commence par décompter le nombre de phrases dans ce corpus, ainsi que le nombre de phrases dans lesquelles le mot est trouvé. Le nombre de phrases contenant le mot est ensuite divisé par le nombre de phrases au total, puis la formule mathématique du logarithme est appliqué au résultat obtenu. Le TF-IDF multiplie donc la partie du TF à celle de l'IDF, ce qui permet de voir le poids de ce mot dans la page Web. Pour revenir sur notre exemple, voici les résultats obtenus pour le mot "chat" :

TF-IDF(« chat », P1, F) = 0,111 * 0 = 0
TF-IDF(« chat », P2, F) = 0,181 * 0 = 0

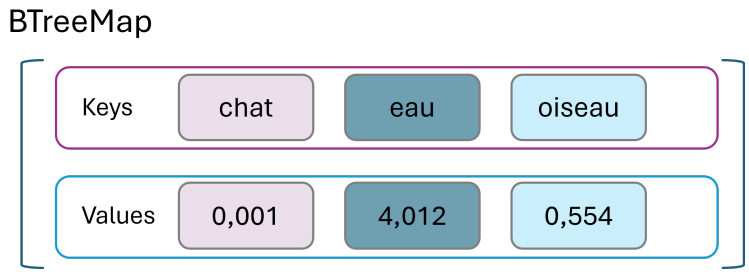
Plus un résultat TF-IDF est élevé, plus cela signifie que le mot est rare et donc potentiellement un mot clé. L'algorithme du TF-IDF est très pratique pour notre projet or sachant qu'il faut parcourir plusieurs fois une même page Web pour obtenir toutes ces variables, le programme peut être assez lent.

## 5.7 Optimisation TF-IDF — BTreeMap

Comme mentionné précédemment, l'algorithme du TF-IDF peut être assez lent, surtout si l'on décide de garder en mémoire tous les résultats obtenus. La méthode la plus naïve pour appliquer le TF-IDF consiste à parcourir plusieurs fois le document, en gardant dans des Vecteurs ou des Arrays les nombres d'occurrences de chaque mot, les différents phrases du texte etc. Même en arrivant à tout faire en un seul parcours, cela reste peu optimisé et inexploitable. Pour éviter cela, il est donc nécessaire d'utiliser des structures de données optimisées permettant de stocker les résultats tout en étant plus rapide qu'un simple vecteur. En Rust, il existe trois grandes structures de données aidant à réaliser ceci :

— les HashMaps,
— les HashSets,
— et les BTreeMaps.

Les HashSets, bien que très pratique pour leur rapidité de recherche et leur efficacité, ne sont pas utiles dans le cas du TF-IDF car ils ne peuvent stocker qu'une seule valeur et l'accès aux éléments est imprévisible avec des modèles d'itération (boucles for, etc).
Les HashMaps sont des dictionnaires facilement utilisables qui possèdent également une rapidité de recherche remarquable. Or, ces structures ne sont pas ordonnées et peuvent performer médiocrement dans certaines situations allant jusqu'à une complexité de $O(n)$.
La dernière option possible est la structure de données BTreeMaps. Cette dernière permet non seulement de conserver plusieurs valeurs, mais reste ordonnée. Sa complexité est de $O(log(n))$, ce qui représente plus que la complexité des HashMaps dans les meilleurs cas ($O(1)$), or les BTreeMaps restent stables et ne dépassent pas $O(log(n))$ même dans les pires situations, ce qui est très pratique pour le TF-IDF qui parcourt des pages Web pouvant posséder jusqu'à plusieurs dizaines de milliers de mots.




Nous utilisons donc des BTreeMaps afin d'optimiser le TF-IDF : ces derniers nous permettent de garder en mémoire le mot clé en tant que "Key" ainsi que son résultat TF-IDF en tant que "Value". Les BtreeMaps sont donc utilisés comme des dictionnaires ordonnés et optimisés. Malgré les optimisations, le TF-IDF reste énorme au niveau de la quantité de données à traiter, ce qui peut (en fonction de la taille de la page Web) rendre la fonction un peu longue. Malheureusement ceci reste un des problèmes de l'algorithme du TF-IDF que nous ne pourrons pas résoudre. En effet le TF-IDF reste un algorithme possédant une approche primitive qui ne prend pas en compte certains facteurs tels que les synonymes, les intentions de recherches ainsi que les objectifs de rédaction.

## 5.8 Récupération des mots intéréssants

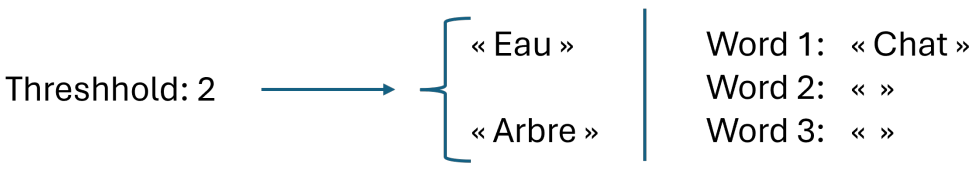
Grâce à l'implémentation de l'algorithme du TF-IDF, la récupération des mots pertinents et intéressants qui décrivent notre site Web est devenue beaucoup plus simple. Une fois le BTreeMap<String, f64> récupéré, il faut le transformer en vecteur. Or une fois devenu un vecteur de tuples (tuples composés de String et d'un f64), l'ordre des mots n'est pas encore établit et il faut donc les trier. Il est donc nécessaire de trier ce vecteur en ordre croissant afin d'avoir les mots peu pertinents au début suivis des mots importants. On indique un "threshhold", soit une valeur seuil qui nous indiquera quand nous arrêter lors du recueil des mots, puis On récupère par la suite les mots dont le TF-IDF se rapproche le plus de 1, signifiant qu'ils sont utiles et appropriés pour décrire ce site Web.

Il est possible en récupérant ces mots que certains d'entre-eux soient des déterminants comme "des", "la", "à". Ces derniers peuvent complètement fausser la base de données vu qu'on les retrouve dans tous les sites Web français, il faut donc les supprimer de la liste. Pour cela, nous avons mis en place un dictionnaire comprenant la majorité des déterminants et articles afin de les éviter par la suite. Il a aussi été nécessaire de le faire en anglais, or sachant que cela reste un dictionnaire de taille conséquente implémenté à la main, nous n'avons pas pu le faire pour toutes les langues.

De plus, nous avons pris les 3 mots les plus pertinents après la valeur seuil s'ils existent, sinon nous prenons des String vides. Cela sert lors de la recherche des sites Web qui correspondent à la demande de l'utilisateur ainsi que pour la propositions de sites Web similaires. En effet il suffit de re-parcourir la base de données et prendre quelques sites Web correspondant à un des trois mots


[(« Chat », 0,001), (« Eau », 0,2), (« Arbre », 0,014), (« Le », 0,34)]

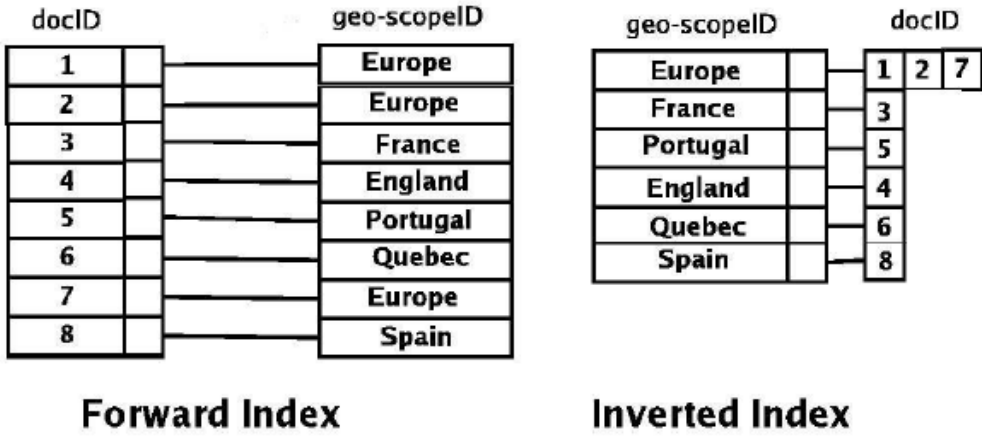
[(« Chat », 0,001), (« Arbre », 0,014), (« Eau », 0,2), (« Le », 0,34)]



## 5.9 Base de données

### 5.9.1 Inverted Index

Les bases de données utilisant un "Inverted Index", autrement appelée index inversé, sont une structure de base de données essentielles pour les systèmes de recherche d'informations. Dans une base de données traditionnelle utilisant une structure en "Foward Index", les données sont archivées sous forme de tableau, où chaque ligne représente un enregistrement et chaque colonne représente un domaine. Dans le cas d'un moteur de recherche, des critères tels que des mots-clés et le PageRank, ainsi que des URLs de site Web sont enregistrés. A chaque entrée, soit des URLs, un ensemble de mots clés lui sera assigné. Cependant, dans une base de données à index inversé, les URLs des sites Web sont stockées sous la forme d'un ensemble où chaque entrée correspond à un mot clé bien précis dans le texte. Les bases de données




à index inversé sont principalement utilisées lors de traitement textuel. Les bases de données à index inversé présentent plusieurs avantages par rapport aux bases de données à index à termes. Premièrement, ces structures de bases de données sont beaucoup plus rapides lors de recherche de grandes quantités de données textuelles. Étant donné que l'index ne contient que des informations sur les mots du texte et non sur le texte lui-même, cela signifie que cette structure est très compacte et que l'index peut être beaucoup plus petit que les données d'origine. Cela implique que les recherches peuvent être effectuées beaucoup plus rapidement et efficacement, même sur de très grands ensembles de données. Les bases de données à index inversé sont également plus flexibles que les bases de données traditionnelles. Étant donné que l'index est créé sur la base des mots-clés du texte, il peut être utilisé pour rechercher n'importe quel terme ou combinaison de termes, sans avoir besoin de langages de requête complexes. Cela permet de trouver plus facilement des informations liées à la recherche de l'utilisateur.
Or, un inconvénient des structures à index inversé est qu'elles consomment beaucoup d'espace mémoire pour se mettre à jour et enregistrer les informations nécessaires. Selon la taille et la complexité de la collection de documents, l'index inversé peut être plusieurs fois plus volumineux que les documents d'origine et peut croître rapidement à mesure que de nouveaux documents sont ajoutés ou modifiés.

### 5.9.2 Organisation de la base de données

Sachant que la majorité des navigateurs Web utilisent une base de données en structure en Inverted Index pour sa simplicité et son efficacité, nous avons décidé d'utiliser aussi ce format de base de données. La base de données est au coeur de ce projet, car sans cette dernière, les parties du Crawler & Scraper et du User Input Processing ne pourraient pas communiquer. C'est pour cela qu'une organisation nette et efficiente est une nécessité absolue. Nous avons donc choisi de séparer notre base de données en deux tables SQLite.

Pour la première table de notre base de données, nous avons décidé de mettre les sites Web avec leur information. Tout d'abord le site Web possède un ID (qui correspond à la clé primaire de notre table). Une fois qu'un ID lui a été attribué, nous entrons son URL, ainsi que sa langue, sa date, son titre et les trois mots supplémentaires obtenus avec le TF-IDF.

| ID | URL | Langue | Date | Titre | Mot 1 | Mot 2 | Mot 2 |
|---|---|---|---|---|---|---|---|
| 1 | www.SiteWeb.com | fr | 2001 | « Les chats » | « Poils » | « doux » | « chaud » |
| 2 | www.SiteWeb2.com | fr | 2024 | « L'eau » | « rivières » | « mer » | « » |
| 3 | www.SiteWeb3.com | en-US | 2011 | « cats and trees » | « oak » | « fur » | « » |
| 4 | www.SiteWeb4.com | en-UK | 2003 | « cars » | « engine » | « oil » | « movie » |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

Ces informations seront utilisées par la partie du User Input Processing lors de la recherche de sites Web. Pour notre deuxième table de notre base de données, nous avons choisi de faire en sorte qu'il n'y est que deux champs : un mot clé et une liste de IDs de site web. La clé primaire de cette dernière est le mot clé. De plus, nous avons choisi de faire en sorte que la liste d'IDs soit sous forme de chaîne de caractères pour simplifier l'ajout ainsi que la recherche.

| Keyword | IDs |
|---|---|
| « chat » | 1, 7, 8, 11 |
| « cat » | 3, 5 |
| « arbre » | 2, 6, 19, 24 |
| « eau » | 2, 1, 23, 57 |
| | |

### 5.9.3 Problèmes rencontrés

Pour l'implémentation de notre base de données, nous avons fait le choix d'utiliser une librairie externe appelée Diesel. Cette librairie nous permet de faire des requêtes SQLite simplement, ce qui nous a beaucoup aidé pour la partie User Input Processing. Or, cette librairie nous a aussi causé énormément de problèmes lors de l'ajout et de la recherche dans la base de données. Lorsque nous avions implémenté l'organisation de base de données décrite ci-dessus, nous ne pouvions pas rechercher des ID précis dans notre liste d'IDs. De plus, nous avions de gros problèmes de conversion entre certains types ce qui nous empêchait d'utiliser certaines fonctionnalités. Après plusieurs jours à tenter de régler ces problèmes (dont la plupart sont dû à des opérabilités non-implémentées dans la librairie), nous nous sommes rendu compte que nous ne pourrions pas faire en sorte d'avoir l'organisation de base de données voulues et nous avons du repartir sur une organisation plus simple et moins optimisée.

Notre première table reste inchangée, nous avons toujours un ID représentant un site Web, ainsi que l'URL de ce site, sa langue, sa date, son titre et ses trois mots supplémentaires. Or, en ce qui concerne la deuxième table, beaucoup de choses ont du changer.
Premièrement, nous avons du rajouter un indice auto-incrémenté qui ne nous sert que pour la clé primaire de cette table. De plus, nous n'avons pas pu intégré la liste des IDs, donc nous avons choisi de mettre un ID de site Web par mot clé, même si cela signifie que le mot clé devra être répété plusieurs fois.

Voici notre deuxième table après avoir changé l'organisation dû aux problèmes rencontrés avec la librairie Diesel.

| ID | Keyword | ID_Of_Site |
|---|---|---|
| 1 | « chat » | 1 |
| 2 | « chat » | 7 |
| 3 | « chat » | 8 |
| 4 | « cat » | 3 |
| 5 | « cat » | 5 |
| 6 | « arbre » | 2 |
| 7 | « arbre» | 6 |
| 8 | « arbre » | 19 |
| 9 | « eau » | 2 |
| 10 | « eau » | 1 |
| 11 | « eau » | 23 |
| | | |

# 6 Crawler - Léo

Pour le bon fonctionnement de DroXyd, nous devons établir une base de données contenant un maximum de sites webs, mais nous n'allons pas établir cette base de données seuls et à la main. . .Pour cela nous allons utiliser un crawler, un algorithme qui nous permet de parcourir internet comme si on parcourait un graphe.
Pour faire fonctionner un tel algorithme, il faut se poser certaines questions. Tout d'abord, comment parcourir les sites webs, puis comment visiter judicieusement internet, ne pas visiter des sites parlant tous drs fois les mêmes sites.
Nous allons parler de tous ces problèmes et y apporter des solutions dans quelques instants.

## 6.1 Le filtre de Bloom

Le filtre de Bloom et son algorithme forment un outil extrêmement puissant basé sur la probabilité de présence d'un élément dans un ensemble, l'intérêt d'un tel algorithme est de nous permettre de savoir si un site web a déjà été visité et c'est une des méthodes les plus rapides en terme de temps d'éxécution. Cette vérification de présence est extrêmement importante, car visiter un site web plusieurs fois revient ensuite à explorer les sites cachés dans celui-ci plusieurs fois. Voyons en détail son fonctionnement.

### 6.1.1 Fonctionnement du filtre

Le filtre de Bloom en lui-même n'est qu'une simple liste composée de valeurs égales à 0 ou 1. Premièrement, l'entièreté du filtre est initialisée à 0. Soient 3 fonctions de hachage $H_1$, $H_2$ et $H_3$ que nous définirons plus tard, ainsi que $x$, un mot que l'on souhaite ajouter à l'ensemble. C'est à l'aide de ces 3 fonctions que nous allons ajouter un élément à notre ensemble.
En supposant que $H_1$, $H_2$ et $H_3 : E \longrightarrow$ [0, Taille de l'ensemble]. Alors, nous allons modifier les valeurs situées aux emplacements $H_1(x)$, $H_2(x)$ et $H_3(x)$ du filtre par 1.
L'objectif étant d'appliquer ce processus à tous les éléments que nous comptons faire appartenir à l'ensemble.
La question qui se pose naturellement est donc : Comment savoir si l'élément recherché est présent dans l'ensemble ?
Pour cela, il va falloir appliquer nos 3 fonctions de hachage au mot $y$ que l'on recherche. Les valeurs situées aux emplacements $H_1(y)$, $H_2(y)$ et $H_3(y)$ seront donc égales à 0 ou 1. Si au moins l'une d'entre elles vaut 0, on pourra alors affirmer que l'élément ne fait pas partie de l'ensemble.
En revanche, il est possible que les 3 valeurs valent 1. Mais ce n'est pas pour autant qu'on peut affirmer à 100% que l'élément est présent, en effet, il existe un risque d'obtenir un "Faux négatif", cet évènement peut s'expliquer par le fait que d'autres mots peuvent avoir les mêmes valeurs de hachage.
Dans ce cas, une des solutions les plus efficaces pour remédier à ce problème est d'augmenter la taille du filtre, ce qui peut nous permettre d'éviter certaines collisions. Évidemment il est possible que d'autres collisions apparaissent suite à ce changement mais on partira du principe que la probabilité que ce cas se présente est extrêmement faible.

### 6.1.2 Fonctions de hachage

Nous avons précédemment parlé de fonctions de hachage, pour l'implémentation du filtre de Bloom, nous en utiliserons toujours 3 et ce choix est justifié. Utiliser moins de 3 fonctions de hachage résulte à une fréquence beaucoup trop élevée de résultats "Faux positifs" donnés, puis, en utiliser plus de 3 nous amène à un nombre de collisions trop élevé, et pour éviter ces collisions, il faudrait augmenter la taille du filtre, qui, en se projetant dans le futur, sera déjà assez grand comme cela.
Parlons maintenant un peu plus en détail de ces 3 fonctions :

— Nous définirons $H_1(x)$ comme la fonction de hachage cryptographique "$MD_5$", autrefois utilisé pour calculer l'empreinte numérique d'un fichier, elle s'avère en réalité très utile pour simplement hacher un mot ou une chaîne de caractères, au vu de la complexité du résultat et du fait que le résultat soit encodé sur 128 bits.

— $H_2(x)$ comme la fonction "SHA-256", cette fonction fait partie d'une "famille" de fonctions de hachage conçue par la NSA, contenant aussi la fonction "SHA-512", celles-ci ont été beaucoup inspirées du modèle de la fonction $MD_5$. Cette fonction, quant à elle, renvoie un résultat encodé sur 256 bits.

La différence flagrante que l'on trouve entre $H_1$ et $H_2$ est la taille du résultat donné par les fonctions (128 contre 256). Mais elles possèdent toutes deux un point commun qui les démarque des simples fonctions de hachage.

```
SHA-256 Tests
DroXyd is the best project made by 2027 students ! = c23132328bf20f1fd17532ad134f12644750f3f2570b32b56d2f6773719d8b4
Droxyd is the best project made by 2027 students ! = af4e0c57b7af23e3a42dd1693893644bc88567cd70c8612bb77bf6c1380fd4e
DroXyd is the best project made by 2028 students ! = ace1b60e9dc4838723967cdc733db5b3576a7a7427957c1de307eb14d66b809
droXyd is the best project made by 2027 students ! = 7d9db59d3b147c7f1f6f74f4be5413be1125a1be2041c840cb7f93ef15c66f7c
droxyd is the best project made by 2027 students ! = 281bcae044c34723e228e05778ef72dafa55f1a96c91d6265eb472bf961e93b1
DroXyd is the best project made by 2028 students !! = e65993a7da42f71c9a5dff76f256336a19a554fd792537f2e6e66dceb468cb1d
```

FIGURE 1 – SHA-256 : Plusieurs tests réalisés sur des strings presque similaires

```
MD5 Tests
DroXyd is the best project made by 2027 students ! = b6d843baf95d68425544a99b72671112f
Droxyd is the best project made by 2027 students ! = 2ad15bbacd5680422843b1b7c66a292f
DroXyd is the best project made by 2028 students ! = 778073bae30598421df2c9b7c819412f
droXyd is the best project made by 2027 students ! = b6d843baf95d68425544a99b72671112f
droxyd is the best project made by 2027 students ! = 2ad15bbacd5680422843b1b7c66a292f
DroXyd is the best project made by 2028 students !! = 7f06f3ba288c1842637949b7d9fc12f
```

FIGURE 2 – $MD_5$ : Plusieurs tests réalisés sur des strings presque similaires

— Finalement $H_3(x)$ sera définie comme étant équivalente à $H_2(H_2(x))$, cette fonction est donc une sorte de (SHA-256)$^2$, qui donne des résultats très convaincants. Cette fonction renverra un résultat encodé sur 256 bits elle aussi, et ce résultat sera très différent de celui renvoyé par $H_2$.

### 6.1.3 Exemple

Posons notre ensemble de départ comme étant composé des mots suivants :
Mickey, Minnie, Donald, Daisy, Dingo et Pluto !


FIGURE 3 – Application de chacune des fonctions sur chacun des mots

Après insertion de tous ces éléments, le filtre de Bloom sera mis à jour.

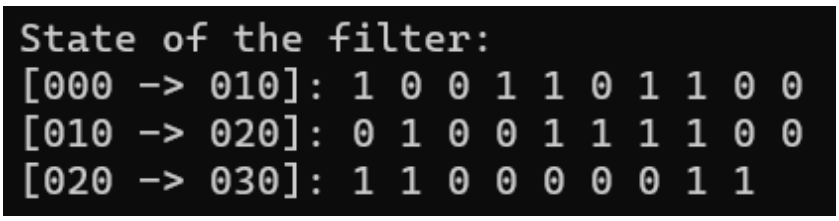



FIGURE 4 – État du filtre après avoir y avoir ajouté tous les mots

La première remarque que l'on pourrait se faire serait d'apercevoir qu'il y a au total 14 / 29 emplacements du filtre occupés par des 1, alors qu'il y avait au total 6 mots à hacher par 3 fonctions chacun, on compte donc parmi tous nos mots des mots qui possèdent des valeurs de hachage égales, ce qui nous arrange car moins on trouve de valeurs fixées à 1 dans le filtre, plus on sera précis !
Il nous reste maintenant plus qu'à regarder si les tests de présence sont vérifiés !

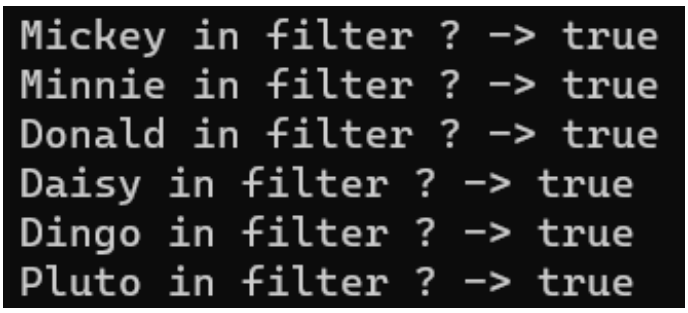



FIGURE 5 – Vérification des mots présents dans l'ensemble de départ

Puis, nous allons pimenter les tests de présence en comptant des mots qui ne sont pas présents dans l'ensemble de départ

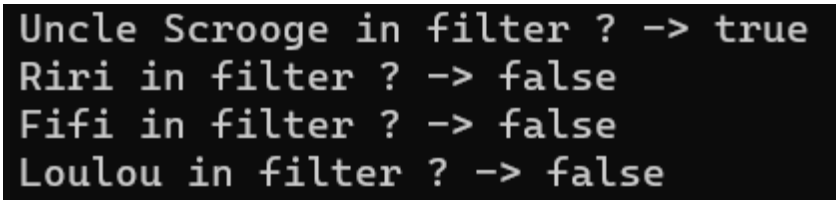



FIGURE 6 – Vérification de mots non présents dans l'ensemble de départ

Il était évident que Riri, Fifi, et Loulou ne faisaient pas partie de l'ensemble de départ, et le filtre l'a bien compris. En revanche, il n'en est pas de même pour l'Oncle Picsou ! Qui est quant à lui indiqué comme présent dans l'ensemble, c'est donc un cas de "Faux négatif".
Il faut tout de même garder à l'esprit que le filtre de Bloom nous permet d'éviter le plus possible de faire des recherches inutiles dans notre base de données, mais il existe tout de même des cas qui nous obligeront à nous assurer que l'élément n'est pas dans l'ensemble dans le cas d'un "Faux négatif".

## 6.2 Récolte et indexation des sites webs

Pour que l'on puisse effectuer des recherches sur le web, il est important d'avoir des sites web à indexer ! Pour cela, il faut donc récupérer leurs adresses, ainsi que leur contenu.
C'est ainsi que l'on trouve une utilité au "crawler", l'objectif est simple, en partant d'un site, on récupère tous les sites qui y sont référencés pour les visiter puis on répète ce processus autant de fois que l'on veut.

### 6.2.1 Créer une requête HTTP

Comme vu précédemment dans des travaux pratiques effectués durant le semestre dernier, on sait qu'accéder à un site web ne se fait pas en un clic...
En effet, on ne peut pas simplement envoyer l'adresse d'un site web au serveur et espérer qu'il nous renvoie le contenu de celui-ci.
Voici à quoi une requête HTTP est censée ressembler :

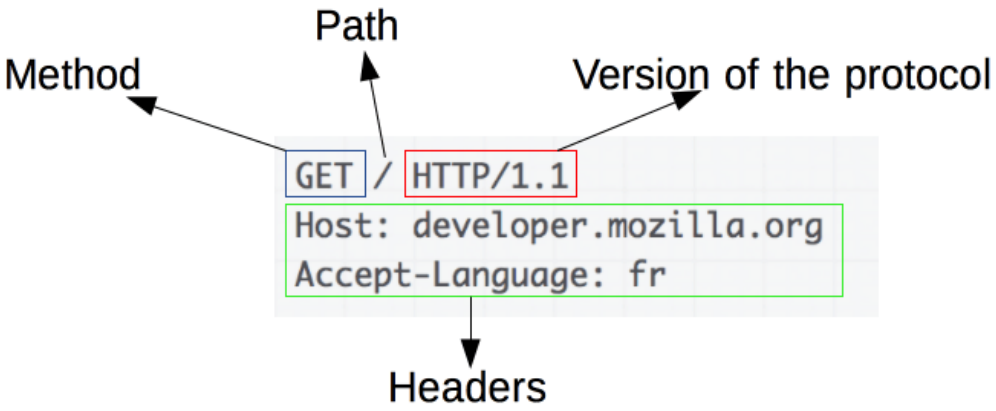



FIGURE 7 – Exemple de requête HTTP

On remarque qu'une très courte chaîne de caractères contient une grande quantité d'informations. Pour nous, la méthode sera toujours la même, car la seule communication que nous voulons effectuer est simplement de récupérer le contenu des sites web, nous n'avons rien à envoyer aux serveurs. On se servira donc seulement de la méthode GET.
Il est aussi possible de rajouter des headers, mais qui, pour notre cas, ne seront pas utiles, la plupart des headers doivent contenir des informations qui nous feraient perdre en optimisation.
On peut illustrer cet exemple avec le header "Date", qui permet d'indiquer dans la requête, l'heure à laquelle celle-ci a été écrite, il faudrait donc obtenir l'heure de la machine pour en faire une version formatée qui, au final, n'apportera rien à la requête.
On retrouve le même cas de figure pour "Expires", qui au lieu d'indiquer l'heure à laquelle la requête a été effectuée, nous indique quand la requête sera "périmée".

### 6.2.2 Communiquer avec le serveur

Maintenant que la requête est créée, il faut l'envoyer au serveur pour récupérer les informations si précieuses que contiennent Internet.
Pour cela, il a été préférable d'utiliser une librairie qui permet d'envoyer la requête ainsi que de récupérer

le résultat. Cette librairie est la librairie "Reqwest" qui comme son nom l'indique, peut traiter des requêtes internet, ainsi, une méthode *send()* nous est fournie et nous permet de communiquer avec le serveur, ce qui nous sera très utile pour récupérer le code source de chacun des sites.

### 6.2.3 "Seeding" ou ensemencement du crawler

Pour que le crawler fonctionne bien et soit amené à explorer une multitude de variétés de sites, il lui faut partir de sites web pertinents et qui donnent accès à plusieurs sites web qui seront eux aussi capable de reproduire le même processus.
Il faut savoir que ces sites seront la base de la recherche de notre crawler, il faut donc les choisir avec précisions tout en étant certains qu'ils aient le moins de liens possibles entre eux pour éviter que deux sites "source" se retrouvent trop rapidement avec les mêmes références.
On peut se représenter cette situation simplement en partant de deux "sujets" différents mais qui traitent d'un même domaine comme le théorème de Pythagore et le théorème de Thalès autour des mathématiques. Tout en restant sur Wikipédia, on peut, en moins de 5 clics, passer d'une page à l'autre. Ainsi, on comprend que plus deux sujets seront éloignés les uns des autres, moins ils auront de chances de contenir les mêmes hyperliens.
Nous avons donc choisi comme sites web de départ les sites suivants :

- La page wikipédia du nombre 42 en anglais
- Un article scientifique trouvé dans le journal "Le Point" sur les éclipses solaires
- Un article philosophique axé sur la question : "L'argent fait-il le bonheur ?"
- La page web de la coupe du monde de football trouvée sur le site de "L'équipe"
- Une question autour des web crawlers posée sur le site StackOverflow

### 6.2.4 Expansion du crawler

En ce qui concerne l'expansion du crawler, nous avons décidé de l'implémenter tel un parcours en profondeur de graphe.
L'utilisation des files nous simplifie grandement la tâche, nous l'initialisons comme contenant les liens de nos 5 sites de base. Nous prenons ensuite chaque élément de la file (dans l'ordre) pour y ajouter tous les sites "voisins", et répéter le processus jusqu'à vider la file.
Évidemment, pour des raisons simples de stockage, de patience et de sécurité, nous n'allons pas naviguer dans l'entièreté d'Internet. Nous pouvons définir une limite au crawler, qui, au-delà de cette limite, arrêtera de parcourir les sites web et stoppera son processus. Une sorte de sécurité pour ne pas avoir à visiter les millions de sites web disponibles.
L'avantage d'avoir créé le filtre de Bloom dont nous avons parlé lors de la soutenance précédente est que les expansions vers les sites web déjà visitée n'est pas effectuée, et l'algorithme du filtre du Bloom nous permet de vérifier cette présence ou non extrêmement rapidement.
Malheureusement, la contrainte à cela est que si un élément est détecté comme déjà visité, nous prenons le risque de ne pas faire de vraie vérification pour s'assurer à 100% qu'il n'a pas déjà été visité car cela prendrait trop de temps, il est donc possible que certains sites ne soient pas visités.

## 6.3 Optimisation du crawler

Avant de débuter cette section, il est important de préciser que pour la partie suivante, le terme "optimisation" ne désigne pas seulement une réduction du temps d'exécution ou une réduction du coût en ressources, mais aussi de la qualité du crawl.

### 6.3.1 Inclure toutes les langues

Pour commencer, si on cherche à exporter DroXyd à l'international, on ne pourra pas se contenter d'avoir visité tous les sites web français, il faudra s'ouvrir au monde, et donc, pour cela, nous devons explorer un maximum de sites écrits dans des langues étrangères au français. Seulement, en partant de sites entièrement français, la probabilité d'explorer des sites anglophones ou autres est minime, en effet, un site de langue étrangère sera rarement référencé sur un site français. La plupart du temps, lorsque qu'une exception vient nous prouver le contraire, ce sera à propos de sujets trop "niches" ou de travaux scientifiques, qui, quelle que soit la langue, reste compréhensible par l'utilisateur.

Quoi qu'il en soit, l'utilisateur doit tout de même pouvoir faire face à des sites non-francophones, pour cela, on peut modifier notre liste de liens de départ, maintenant, celui-ci sera composé des sites suivants :

— DMOZ, un site contenant une quantité phénoménale d'hyperliens ammenant vers des sujets divers et variés.


FIGURE 8 – Le site DMOZ, permettant l'accès à des sites divers et variés

— La page wikipédia du nombre 42 en anglais.
— Un article scientifique trouvé dans le journal "Le Point" sur les éclipses solaires.
— Un article philosophique axé sur la question : "L'argent fait-il le bonheur ?".
— La page web de la coupe du monde de football trouvée sur le site de "L'équipe".
— La page wikipédia italienne sur la pizza.
— Le site officiel des jeux olympiques.
— La page centrale du New York Times.
— La page Wikipédia du manga "Attack on Titan" en allemand.

— La page Wikipédia du film "Pulp fiction" en espagnol.

Grâce à cette sélection, on peut être sûrs de tomber sur n'importe quels sites de n'importe quelles langues.

### 6.3.2 Choix des sites avec plus de précaution

Suite à la seconde soutenance, le crawler était capable d'explorer internet, mais il le faisait de manière "stupide" car il explorait des pages chacunes dans l'ordre, ce qui pouvait poser un problème, un cruel manque de diversité. Par exemple, la page d'accueil de Wikipédia contient une quantité d'hyperliens ahurissante, mais aussi plusieurs liens qui ne nous serviront pas, notamment des redirections vers des sites partenaires de Wikipédia ou des sites contenant une partie de la base de données du site. On ne trouvera rien sur ces sites et on sait que ces sites ne sont pas référencés par nos moteurs de recherche préférés.

Mais attention, il y a aussi des sites cherchant à tout prix à être référencé par les moteurs de recherche, ces sites utilisent une méthode appelée "Search Engine Optimization" ou SEO. Leur objectif est simplement d'utiliser un maximum de stratégies de référencement et des les combiner pour obtenir un rang de page élevé. Ainsi, certaines pages sur Internet possèdent un nombre incalculables de liens externes. Bien heureusement, nous avons peu de chances de trouver leurs adresses puisqu'ils ne sont référencés sur presque aucun site, personne ne veut référencer des sites d'une aussi mauvaise qualité. Seulement, il existe même des pages Wikipédia qui utilisent un tel schéma, notamment la page du nombre 42, en effet, cette page possède un lien externe pour une très grande partie des nombres compris entre 1 et 1000.

| Liste de nombres voisins | [masquer] |
|---|---|
| Unités voisines | ← 40 · 41 · **42** · 43 · 44 · 45 · 46 · 47 · 48 · 49 → |
| Dizaines voisines | ← 0 · 10 · 20 · 30 · 40 · 50 · 60 · 70 · 80 · 90 → |
| Centaines voisines | ← 0 · 100 · 200 · 300 · 400 · 500 · 600 · 700 · 800 · 900 → |
| Milliers voisins | ← 0 · 1000 · 2000 · 3000 · 4000 · 5000 · 6000 · 7000 · 8000 · 9000 → |
| Dizaines de milliers | ← 0 · 10000 · 20000 · 30000 · 40000 · 50000 · 60000 · 70000 · 80000 · 90000 → |
| Centaines de milliers | ← 0 · 100000 · 200000 · 300000 · 400000 · 500000 · 600000 · 700000 · 800000 · 900000 → |
| Millions voisins | ← 0 · 1000000 · 2000000 · 3000000 · 4000000 · 5000000 · 6000000 · 7000000 · 8000000 · 9000000 → |
| Milliards voisins | ← 0 · 1000000000 · 2000000000 · 3000000000 · 4000000000 · 5000000000 · 6000000000 · 7000000000 · 8000000000 · 9000000000 → |
| Ordres de grandeur | |

FIGURE 9 – Liste de liens externes contenus dans le site du nombre 42

La meilleure solution pour éviter de faire affaire avec ces sites serait de ne pas les prendre en compte, mais malheureusement, on ne peut pas savoir à l'avance si un site fait partie de ces cas particuliers ou non, dans ce cas, la meilleure manière de procéder est de sélectionner des sites au hasard et de réduire un maximum la probabilité de piocher un site "inutile" lors de la prochaine étape du processus, ce qui pourrait aussi permettre un de compléter un catalogue de domaines encore plus garni.

### 6.3.3 Multiple core multi-threading programming

Une optimisation des ressources des plus importantes serait celle du mutiple core multi-threading programming, une notion que nous avons étudié autour de la fin du troisième semestre en C. Il s'avère qu'il serait possible d'utiliser cette particularité en Rust, ce qui pourrait grandement réduire le temps d'exécution du crawler.
Mais à quoi sert le multi-threading ? C'est un processus qui permet à notre code de s'éxecuter simultanément sur un seul processeur.

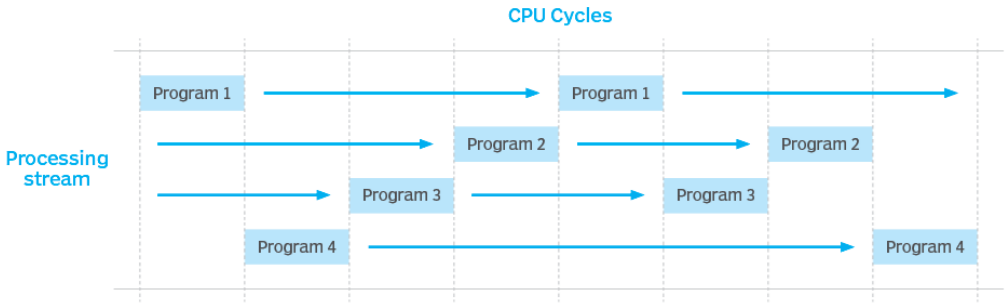



FIGURE 10 – Le multi-threading appliqué sur 4 programmes différents.

Le multi-threading est donc capable de séparer un programme principal en plusieurs sous-parties, puis d'attribuer la puissance et les ressources du processeur à chacune des parties de manière alternée. Maintenant, on peut lier ce concept au multiple core programming, donc, appliquer le multi-threading sur plusieurs coeurs de processeurs. Grâce à cela, on obtient une vitesse d'exécution beaucoup plus rapide qu'habituellement puisque le code est capable de s'exécuter $n$ fois plus vite qu'auparavant, avec $n$ représentant le nombre de coeurs que possède le processeur de la machine. C'est donc une notion extrêmement importante qui mérite d'être utilisée dans le projet pour gagner un temps considérable pendant l'alimentation de la base de données.
Pour ce qui est de l'utilisation du multi-threading, on va diviser les opérations du crawler en plusieurs parties qui fonctionneront en même temps, en effet, lorsque que l'on traite un site web, on va ajouter chacun des sites web trouvés à une liste qu

# 7 User Input Processing

## 7.1 Parser/Lexer - Quentin

Dans cette partie du projet, nous nous sommes concentrés sur la création de la partie user input.
Où en premier partir nous avons créé un parseur lexeur. L'objectif principal d'un analyseur lexical et d'un parseur est de comprendre la structure d'un texte ou d'une phrase donnée, en identifiant et en extrayant les éléments clés qui la composent.
Dans le cadre spécifique de notre lexer/parser, notre objectif est de comprendre précisément ce que l'utilisateur recherche. Pour éviter tout confusion et pour éviter problème pas le suite nous devons bien comprendre ce que l'utilisateur recherche. De plus ce parser/lexer permet aussi au développeur car pour tout les autres parti de l'user input nous avons besoins de connaître ce qu'écrit l'utilisateur. Pour ce faire, nous avons défini plusieurs mots-clés qui permettent à l'utilisateur de spécifier sa requête de manière plus précise et efficace.
Le premier mot-clé est la possibilité de ne rien spécifier, ce qui correspond à une requête normale. Par exemple, une requête comme "pizza fromage" permettrait de trouver tous les sites contenant les mots "pizza" et "fromage" dans leur contenu. Ce type de requête est le plus généralement utilisé pas les utilisateurs car c'est la plus simple et efficace pour faire une recherche car elle ne nécessite pas beaucoup de connaissance et est efficace en toute circonstance.

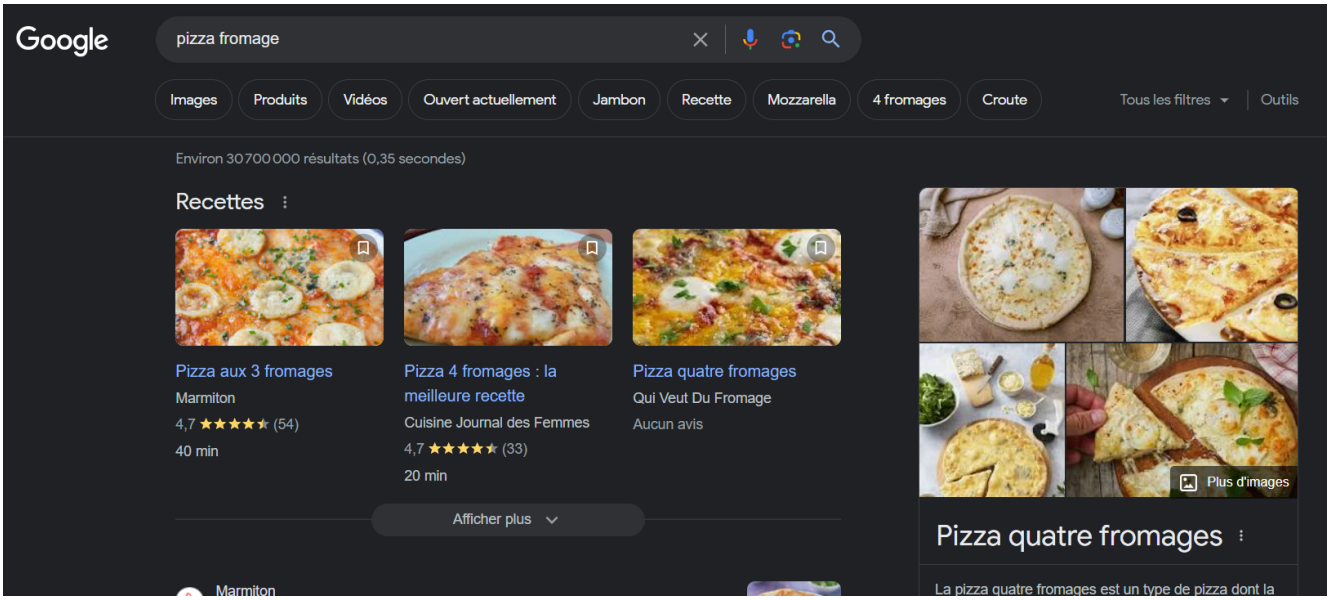



FIGURE 11 – search 1

Un autre mot-clé important est le symbole "-", qui est utilisé pour exclure certains termes de la recherche. Par exemple, en saisissant "-ananas", l'utilisateur exclue tous les sites contenant le mot "ananas" dans leur contenu.

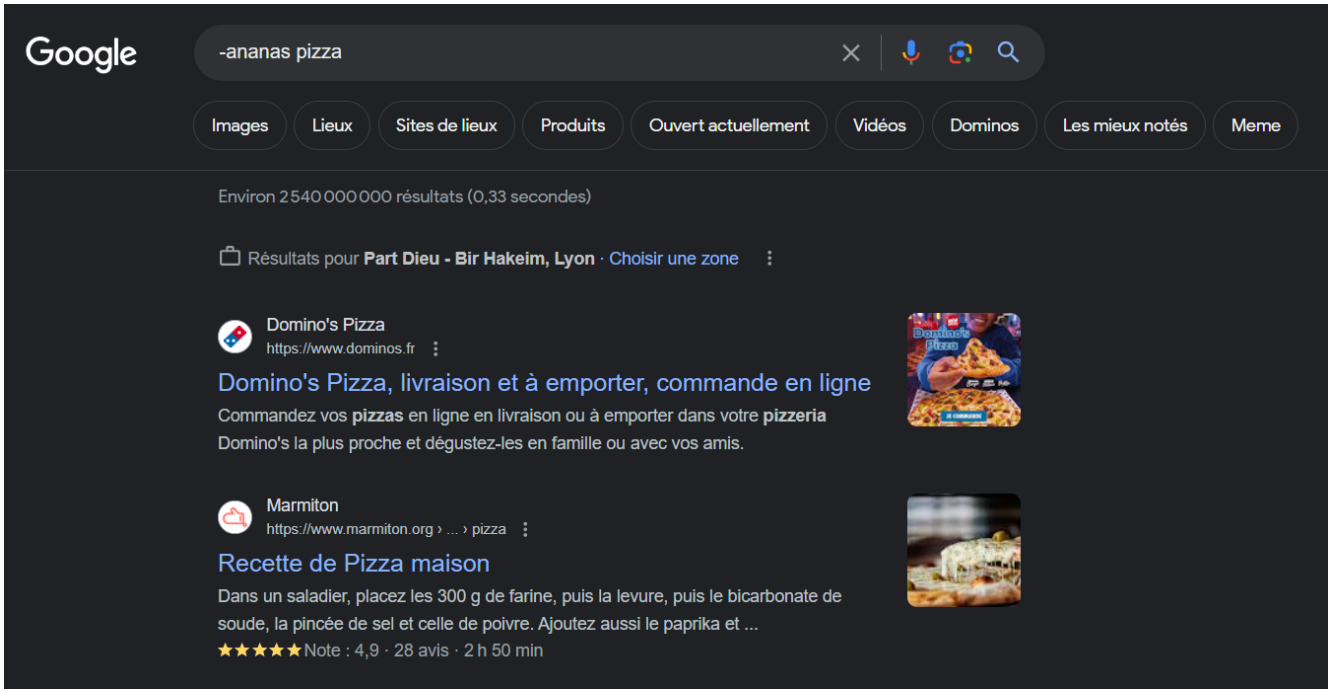



FIGURE 12 – search 2

Bien sûr il en existe d'autre de différente utilité. Enfin, l'utilisation de guillemets (« ») permet à l'utilisateur de rechercher une phrase exacte dans les résultats. Par exemple, « pizza 4 fromage » retournerait des sites contenant exactement cette phrase. En combinant ces différents mots-clés, l'utilisateur peut affiner sa recherche pour obtenir des résultats plus pertinents. Par exemple, en saisissant "fromage site : pizzahut.com unsite : dominos.com -ananas", l'utilisateur exclurait les sites de dominos.com, rechercherait spécifiquement sur pizzahut.com et exclurait tous les résultats contenant le mot "ananas".
Ces fonctionnalités offrent à l'utilisateur la possibilité de mener des recherches précises et ciblées, lui permettant ainsi d'obtenir les résultats les plus pertinents en fonction de ses besoins spécifiques.

## 7.2 SQL query - Quentin

Dans cette Deuxième parti de la user input , nous nous sommes concentrés sur l'utilisation de Rust pour exécuter des requêtes SQL afin d'interroger une base de données.
Nous avons rencontré plusieur problème donc ce processus n'a pas été simple, notamment lors de l'installation des différentes bibliothèques et outils nécessaires à notre tâche car pour installer diesel il nous faillais au moins une librairie SQL et que le premier problème c est retrouvé ici car pour des raison que nous connaissons pas, il nous a fallu installer chocolatey pour pouvoir installer sqlite qui lui un fois installer nous permettais d'installer diesel grace encore a chocolatey.
Donc ce qui a poser des problèmes et du temps perdu car cela a pris une à deux semaine a comprendre comment les installer et donc nous avons choisi d'utiliser SQLite et Diesel pour leur simplicité et leur efficacité, car ils nous permettaient de créer des bases de données, de chercher dans celle-ci et même de modifier ces bases de données avec des fonctions plutôt simple une fois comprise. Donc le problème poser était de comprendre ces fonctions en l absence d exemple et la documentation très peu remplie car rust est un langage qui est utilisé depuis peu et par moins de monde du coup, ce qui donne une documentation réduite plus les librairies peu utilisé donc presque non documenté.
Pour ce qui est de la parti sql nous utilisons une structure pour notre base de données qui est composé de deux table, la première est composé de l'id, url, nom, date, langue , et de trois mots clé d'un site ce qui permet d avoir toute les informations utiles pour l affichage et la recherche dans la base de donnée, les trois mots clé sont la pour dire quelle sont les mots les plus importants du site ce qui permet de les triés et aussi de faire des suggestions et la deuxième table comportant les mots clés et l id du site auquel il appartient ce qui nous permet de faire une recherche dans la base de donnée.

L'algorithme que nous avons développé pour cette partie de notre projet peut être décomposé en plusieurs étapes.

Tout d'abord, nous établissons une connexion à notre base de données. Pour pouvoir exécuter des requête SQL sur celle-ci grace a une fonction disponible sur diesel. Ensuite, pour chaque élément de la structure que nous souhaitons traiter, nous recherchons les informations correspondantes dans la première table qui comporte deux éléments le premier correspond aux mots-clés présents dans les sites visités, donc, key, et le deuxième élément est un ID qui permet de savoir quelle URL prendre. Donc grâce a un inner join présent avec diesel nous pouvons avoir accès à ces deux informations, plus un filtre et une sélection nous avons les bons résultats.

| | |
|---|---|
| pizza | 1 |
| pizza | 2 |
| pizza | 3 |
| tacos | 4 |
| cat | 5 |
| dog | 0 |
| cat | 6 |
| cat | 7 |
| cat | 8 |
| dog | 0 |

FIGURE 13 – Table 1

Puis, avec ces informations en main, nous procédons à une recherche dans la deuxième table pour obtenir des données complémentaires ou associées, en utilisant l'ID de l'autre table, ce qui nous permet de récupérer toutes les informations concernant les sites cherchés.

| id | url | langue | name | date |
|---|---|---|---|---|
| Filtre | Filtre | Filtre | Filtre | Filtre |
| 1 | https://fr.wikipedia.org/wiki/Pizza | fr | wikipédia | 09/04/2024 |
| 2 | https://www.dominos.fr | fr | dominos | 09/04/2024 |
| 3 | https://www.pizzahut.fr | fr | pizzahut | 09/04/2024 |
| 4 | https://fr.wikipedia.org/wiki/Taco | fr | wikipédia | 09/04/2024 |
| 5 | https://fr.wikipedia.org/wiki/Chat | fr | wikipédia | 09/04/2024 |
| 6 | https://www.larousse.fr/... | fr | larousse | 09/04/2024 |
| 7 | https://www.woopets.fr/chat/ | fr | woopets | 09/04/2024 |
| 8 | https://www.santevet.com/... | fr | santevet | 09/04/2024 |
| 9 | https://fr.wikipedia.org/wiki/... | fr | wikipédia | 09/04/2024 |
| 10 | https://www.woopets.fr/chien/ | fr | woopets | 09/04/2024 |

FIGURE 14 – Table 2

Puis, une fois que nous avons récupéré les données nécessaires, nous les trions et les filtrons pour ne conserver que celles qui nous intéressent vraiment, en recherchant parmi les résultats de la requête SQL. Enfin, nous renvoyons ces informations pour les utiliser dans la suite de notre application ou de notre système en utilisant une structure qui comporte toute les informations de notre recherche c est a dire l'url, le nom, la langue,les mots clés et autre.

Cette partie nous a permis de créer un système efficace pour chercher dans notre base de données en utilisant Rust.

## 7.3 Interface - Quentin

Pour cette dernière partie, réalisée pour la troisième soutenance, nous nous sommes concentrés principalement sur l'aspect visuel en ajoutant plusieurs fonctionnalités. Tout d'abord, nous allons parler des problèmes rencontrés. Le premier problème fut l'intégration du CSS dans le code Rust. En effet, pour utiliser du CSS avec Rust, il faut recourir à Rocket, qui présente des difficultés d'utilisation, notamment en ce qui concerne la gestion du CSS. Le deuxième problème rencontré concerne les erreurs. Il est en effet difficile de trouver de la documentation sur Rocket et ses erreurs, ce qui complique davantage le développement. En ce qui concerne la partie visuelle, plusieurs changements ont été apportés : Amélioration de l'interface utilisateur : Nous avons retravaillé l'interface pour la rendre plus intuitive et agréable à utiliser. Cela inclut la refonte des boutons, des formulaires et des autres éléments interactifs. Optimisation de l'affichage : Nous avons optimisé l'affichage pour qu'il soit compatible avec différents types d'écrans et de résolutions, garantissant ainsi une meilleure accessibilité.
Les premiers changements sont les boutons :


FIGURE 15 – Button

Nous avons choisis de changer les boutons qui étaient les boutons de base HTML en des boutons plus simple et épuré, donc le bouton de recherche deviens gris avec un changement de couleur au clic, et l ajout des trois nouveaux bouton pour les dictionnaires fait par alexandre on été modifié pour être plus visible et tape a l'oeil en étant bleu foncé sur du noir.

La barre de recherche :

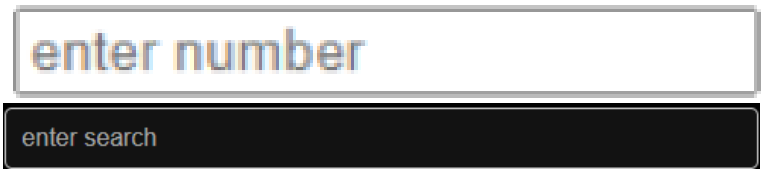



FIGURE 16 – Search Bar

La bar de recherche a été très peu changé car elle était simple et efficace, seul la typographie et le contour on été changée.
Le titre :


FIGURE 17 – Title

Sur le titre seul typographie a été changé.

Sur la deuxième page nous avons ajouté un titre plus un option qui permet de crée une nouvelle page ou de continuer les recherches. L'ajustement des liens pour mieux organisé la recherche.

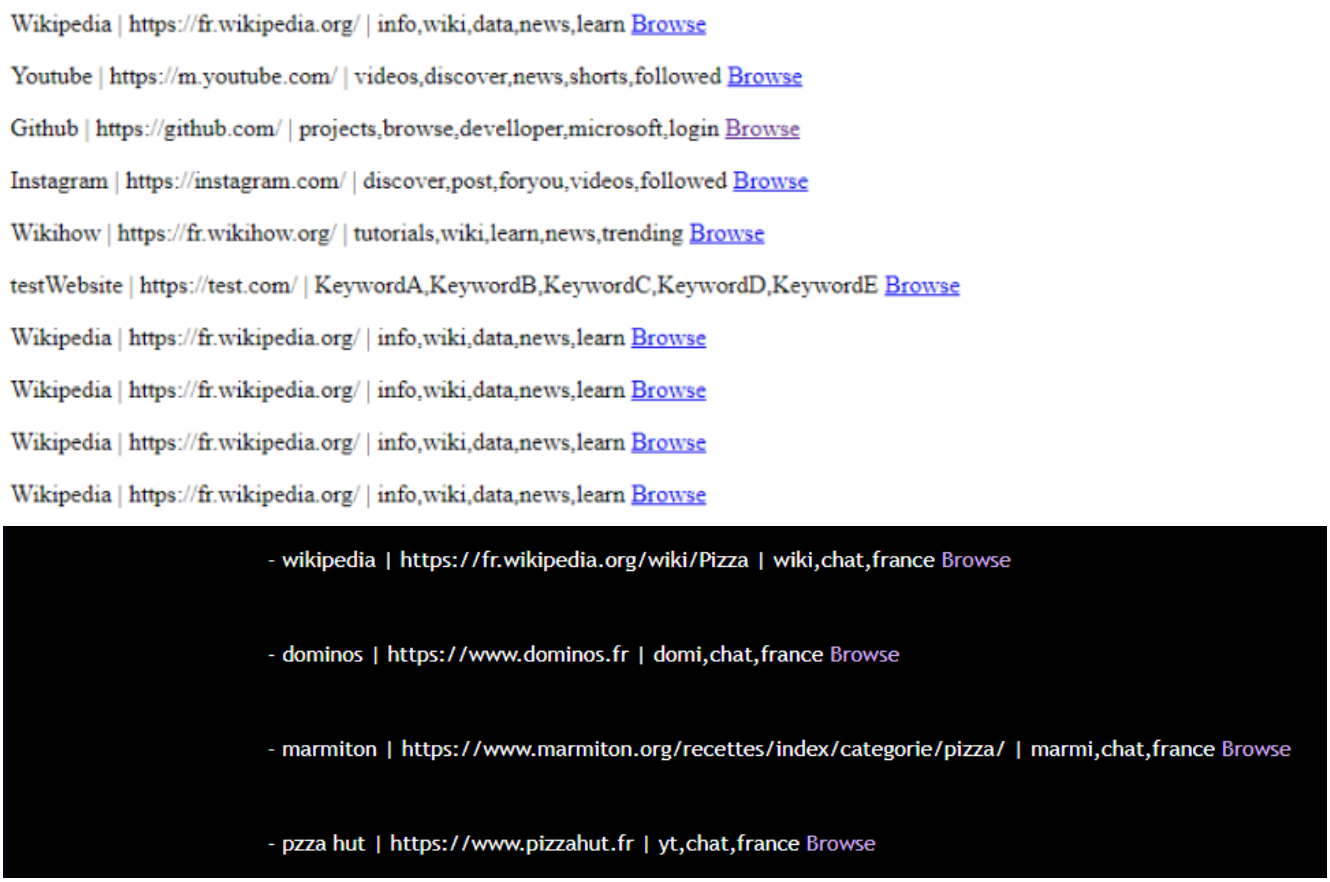



FIGURE 18 – Link

Pour ce qui est de l'affichage des liens seulement un décalage à droite plus un espacement entre chaque lien pour pouvoir avoir une meilleur visibilité et une meilleur lecture des liens plus confortable.

Puis l ajout d'un changement de page qui permet de savoir a quel page nous sommes et permet aussi de changer de page.


FIGURE 19 – Beautiful

Ces changements visent à améliorer l'expérience globale des utilisateurs et à garantir une utilisation fluide et agréable de notre application. Ces changements ont été ajoutés exclusivement pour améliorer l'efficacité de l'utilisation de l'application, tant pour les développeurs que pour les utilisateurs finaux.

Pour les développeurs, ces améliorations permettent de visualiser plus facilement les changements et les modifications apportées au code. L'interface utilisateur améliorée, les nouvelles animations et l'optimisation de l'affichage facilitent le repérage des erreurs et des ajustements nécessaires, rendant le processus de développement plus fluide et efficace. De plus, la documentation et les outils de débogage intégrés sont désormais plus accessibles, ce qui réduit le temps nécessaire pour résoudre les problèmes.

Pour les utilisateurs finaux, ces changements améliorent considérablement l'expérience de navigation. Ils peuvent effectuer des recherches tout en utilisant plusieurs onglets ou en conservant le même onglet ouvert, ce qui leur permet de gérer plus efficacement leurs tâches. L'interface plus intuitive et les options de personnalisation leur offrent une expérience utilisateur plus agréable et adaptée à leurs besoins.

En résumé, ces améliorations visent à rendre l'application plus conviviale et efficace pour tous les utilisateurs, qu'ils soient développeurs ou utilisateurs finaux, en facilitant la navigation, la personnalisation et la gestion des tâches au sein de l'application.
Resultat final :

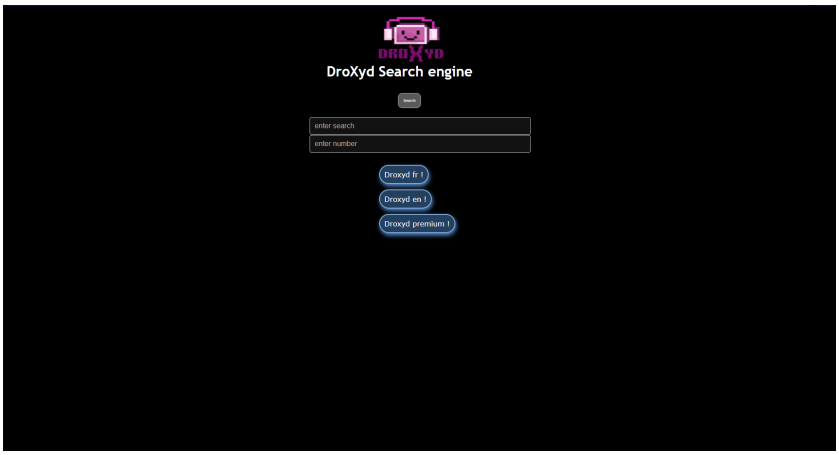



FIGURE 20 – Site

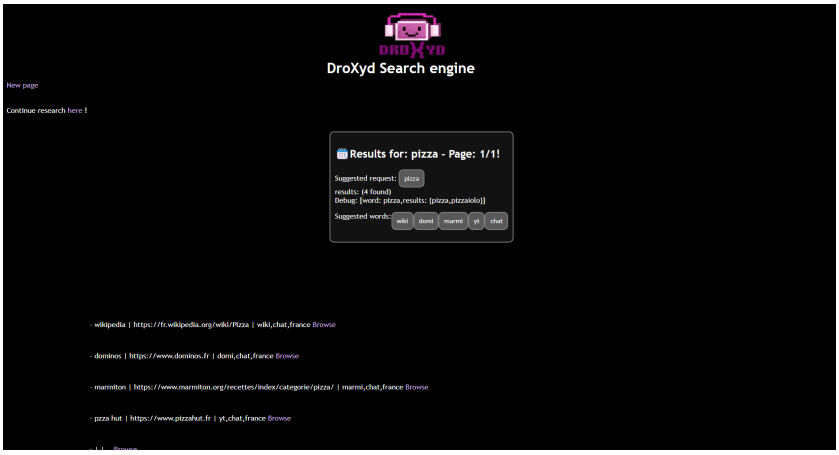



FIGURE 21 – Site

## 7.4 Etat du Correcteur - Alexandre

Le but de cette partie est d'implémenter un pont entre l'utilisateur et le moteur de recherche DroXyd. Il se doit d'être pratique, complet et intuitif. C'est pourquoi ont été implémentées les fonctions de recherche et de correction à une page web qui nous servira d'interface.
Le contenu de cette dernière est divisée en 2 parties. Les améliorations liées aux recherches de corrections, ainsi que le contenu même de l'interface. Tout d'abord, reprenons les fonctionnalités du dictionnaire telles qu'elles l'étaient à la première et 2ème soutenances. On pouvait ajouter des mots, les chercher, les compléter ou les corriger. Il y avait un système de pondération, pour stocker les poids de chaque mots dans le dictionnaire.Enfin, la structure était un arbre ternaire de recherche, la meilleure solution pour stocker et exploiter les données.

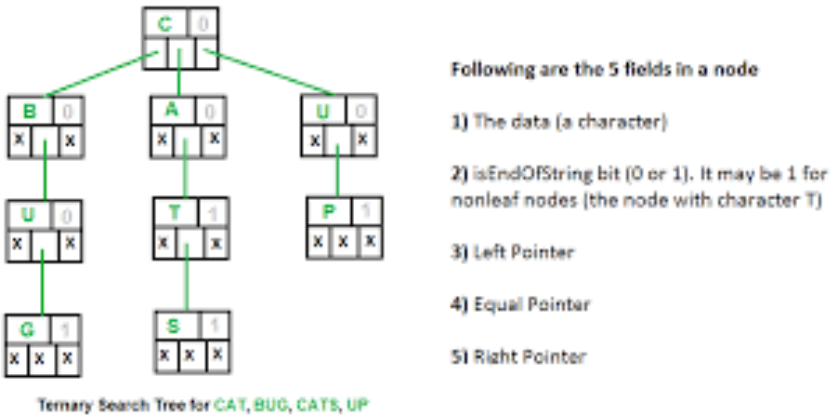



FIGURE 22 – Ternary search tree

Désormais, il est primordial d'adapter le dictionnaire à une utilisation fluide de recherche. Il est capable d'analyser une phrase en entier, d'en dégager les mots les plus importants (en les valuant lors de l'ajout d'un mot dans l'arbre), et de corriger intelligemment les termes dans une phrase. Les mots comprenant des caractères spéciaux, comme "-cat" ou "site :wikipedia" sont traités ultérieurement par le parseur, ainsi ils ne sont donc pas à corriger. Il en va de même pour les dates (ou suites de chiffres), qui sont ignorées lors de la correction mais pas lors de la recherche de résultats.

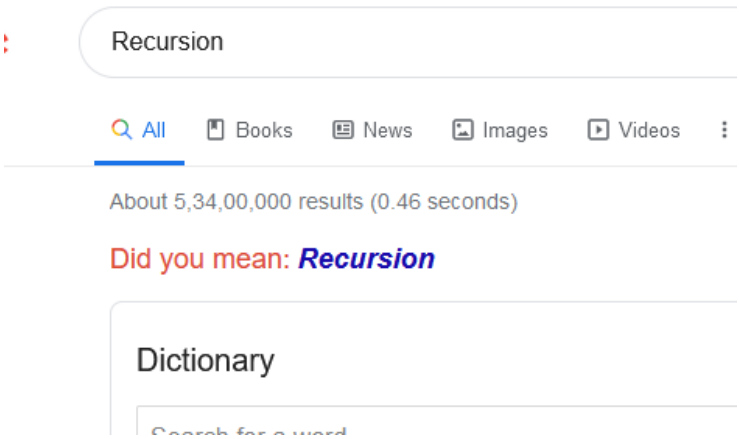



FIGURE 23 – Suggested request

Enfin, les mots mal orthographiés ou incomplet sont revus afin de proposer la meilleure solution. Si le mot peut être complété, alors on prend la suggestion avec la valeur la plus haute sensible de correspondre à l'entrée. Autrement, on choisit la meilleure correction possible, ou dans le pire des cas on supprime l'entrée. Aussi, ont été implémentées les fonctions de suggestions avancées. La requête entière est donc analysée et corrigée, et les résultats de celle-ci sont triés puis affichés. Les mots-clés des résultats les plus courant sont alors mis en avant pour affiner les recherches de l'utilisateur.

Voici quelques exemples :

— On ajoute les mots valués suivants au dictionnaire : ferrari (45), ferry(12), ferraille(3).
— On entre "ferr", et le premier résultat sera alors "ferrari" car il a la valeur la plus élevée (suivi de ferry, puis ferraille)
— On entre "berrari f40 hthhthh", et le résultat suggéré sera "ferrari f40", le premier mot étant corrigé, le second ignoré, et le troisième supprimé. Puisque les mots ferry et ferraille ont une valeur plus basse (car moins présents dans nos bases de données), ils sont moins susceptibles d'être proposés directement. Ils apparaîtront dans les propositions d'auto-complétion les moins pertinentes.

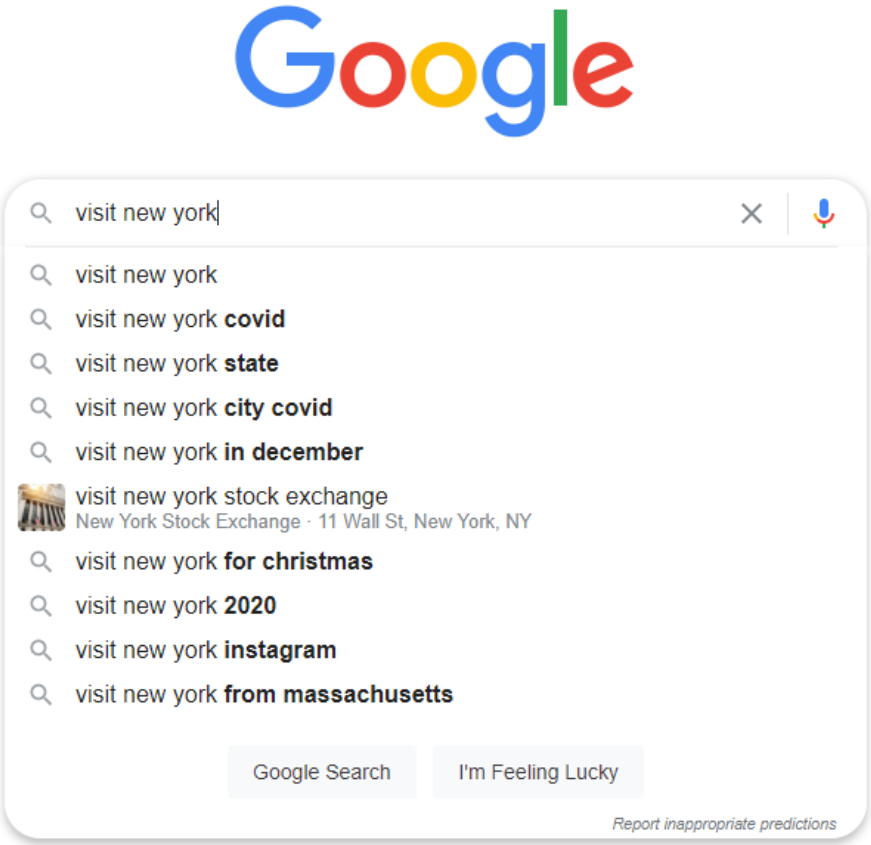



FIGURE 24 – Pondération des résultats

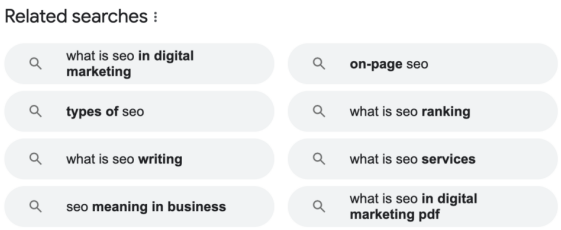



FIGURE 25 – Pondération des résultats

Pour l'interface, il a fallu implémenter une structure pratique afin de simplifier les recherches de l'utilisateur. De ce fait, il est important de créer un environnement propice au développement UI dans le projet. Ainsi en travaillant avec Quentin à la soutenance 3, il a été plus simple de coder sur un support flexible au travail en groupe. C'est pourquoi nous avons utilisé roquet, une librairie de développement d'interface web. Dans l'interface DroXyd Search Engine, il est possible de créer plusieurs tabs de recherche, d'entrer une requête et le nombre de résultats voulus, et de voir dans une nouvelle fenêtre le résultat. Ce dernier est constitué de la requête d'origine, de la requête suggérée, et l'ensemble des résultats du dictionnaire sur les mots entrés. Enfin, les résultats de la requête sont affichés, avec la liste des mots clés suggérés.
Il s'agit là d'un exemple d'affichage du résultat lors de l'appel à la base de données. La liste de résultats est traitée puis les mots clés sont affichés comme tel. Les requêtes précédentes sont sauvegardées dans le navigateur. Aussi, une page web est ouverte au démarrage de l'application =)

Pour finir avec l'interface, j'ai rajouté un champ supplémentaire pour choisir la page de resultat à laquelle on veut aller directement.Par exemple, vous êtes en train de consulter la page 3324 sur les chats et vous fermez votre navigateur. A la prochaine recherche, vous pouvez entrer 3324 dans le second champ pour continuer votre recherche. Aussi, si vous ne rentrez pas de mots clés à votre recherche, vous serez acceuillis par le message suivant :

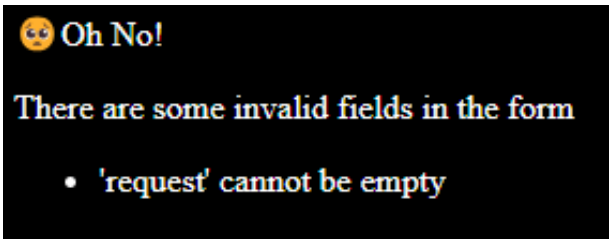



FIGURE 26 – Gestion des erreurs

## 7.5 Better Dictionnaries - Alexandre

Cette partie est marquée par l'implémentation de 150k mots statiques et probablement autant dans la base de données afin de remplir le dictionnaire. Ces mots proviennent des dictionnaires français et anglais, qui sont chargés en mémoire au lancement du programme. Le but est de pouvoir changer de langue ou déterminer si les mots clefs de la base de données pouvaient influer sur les corrections. C'est le départ d'une bataille à l'optimisation pour charger 500k mots le plus vite possible sur des environnements de tests. Les premiers résultats tournaient autour des 4sec, pour enfin converger vers la moitié après de nombreuses optimisations. Il en va de même pour la taille de stockage, que j'ai pu réduire à 1mo pour 150k mots. Tout ça pour dire, que la seule opération qui pouvait nécessiter un écran de chargement au lancement du programme à été optimisée au max. Pour 150k mots, l'opération se fait en 1.2 sec (record) avec 1.3mo de stockage ( au lieu de 12sec et 4mo sur une implémentation avec une liste simple en rust ).

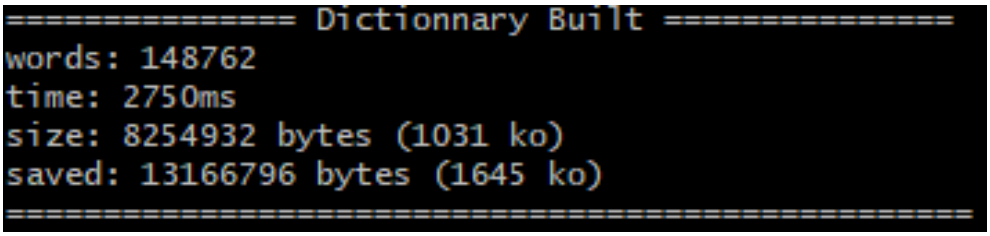



FIGURE 27 – Le dictionnaire

La seconde épreuve à été l'ajout des mots de la base de données au dictionnaire. Il faut faire attention à la taille de la structure, l'orthographe des mots, la langue et le nombre d'apparition. C'est pourquoi j'ai du adapter les fonctions d'ajouts et de recherche pour convenir à ces nouvelles difficultés. Maintenant, la pondération des mots dans le dictionnaire dépend du nombre d'apparition du mot dans la base de données. Pour cela, une requête ( confectionnée par quentin ) récupère la liste des mots, que je recalcule ensuite pour les intégrer au dictionnaire. Enfin, j'ai créé 3 options pour choisir la langue ( français / anglais et premium ) pour offrir une meilleure expérience aux utilisateurs. Français et anglais sont basés uniquement sur les dictionnaires de leurs langues respectives alors que le mode premium charge les 2 dictionnaires + celui de

la base de données ( en recalculant toutes les pondérations ). Ce dernier mode est chargé par défaut au démarrage ( 1.2 sec on le rappelle ahah ). Toutes les pages ouvertes dans votre navigateur seront touchées par ce changement de langue, pratique !


FIGURE 28 – Changer de langues !

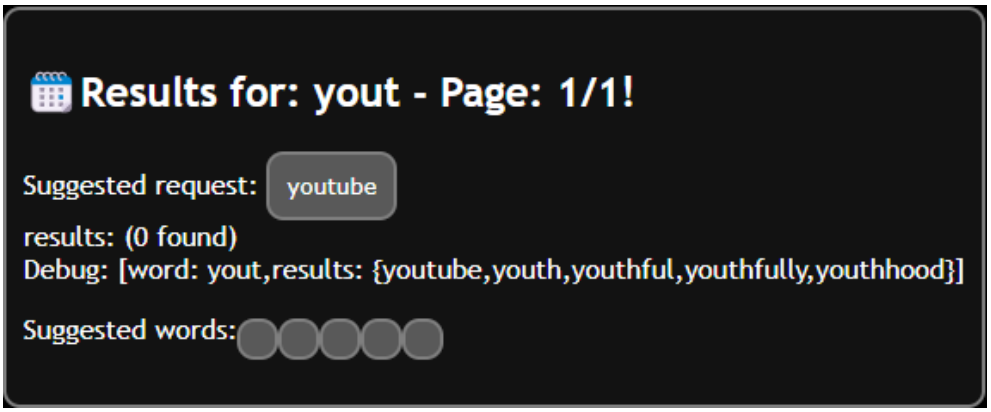



FIGURE 29 – Resultats de "yout"

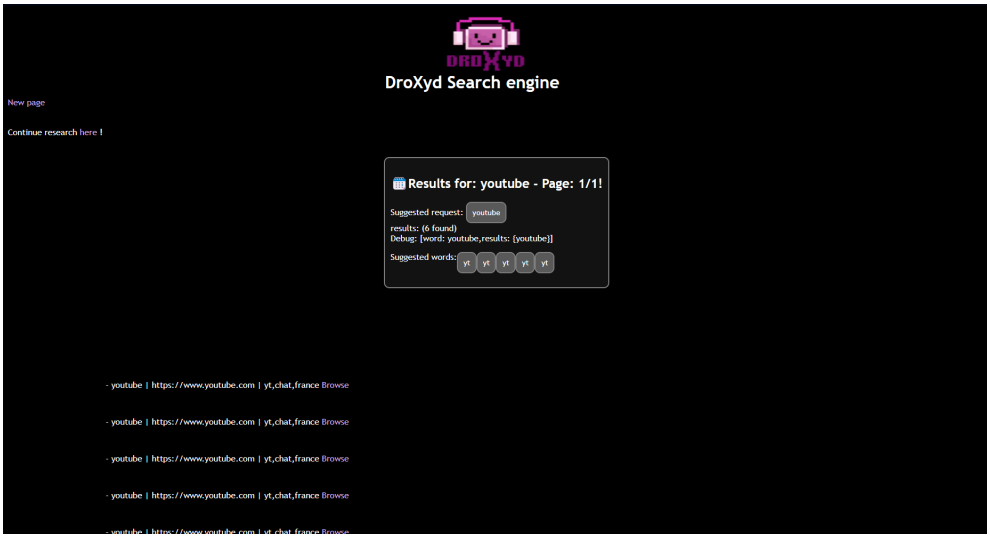



FIGURE 30 – Resultats de "youtube", après avoir cliqué sur le boutton

## 7.6 Autres - Alexandre

Enfin, j'ai pu travailler sur le système de récupération des résultats et leur affichage, le tri, la manipulation des pages et le retour à la page principale. Or tout ceci à été codé avec l'aide de quentin, et nous allons voir ça en détail dans la partie d'après.

## 7.7 Interface - Alexandre & Quentin

Pour commencer, nous avons mit en commun nos travaux sur un seul et même exécutable, pour que les recherches, après avoir été corrigées, puissent être envoyées sous forme de requête à la base de données. Ces informations sont récupérées puis implémentées dans la page de résultats. Lorsque les résultats sont récupérés, ils sont stockés localement sous forme de liste qui doit être triée. Pour trier les résultats, nous utilisons un calcul qui détermine un score pour chaque page, basé sur les recherches précédentes, la quantité de mots clés en communs et les filtres de la recherche. Ensuite, pour naviguer entre les résultats, nous avons ajouté des boutons pour choisir la page de résultats à afficher. Cela semble simple en théorie, mais il a fallu passer par de nombreuses requêtes et canaux complexes entre la page html et le code rust. Le stockage de ces résultats se fait en local pour chaque tab et se fait de manière temporaire. C'est à dire que lorsque la page est fermée, le stockage des résultats est supprimé de l'ordinateur.


FIGURE 31 – Magnifique, n'est ce pas ?

Pour exécuter la requête SQL, nous utilisons la fonction query() que nous avons préalablement créée. Cette fonction est essentielle car elle interagit directement avec notre base de données pour extraire les informations pertinentes. Lorsque l'utilisateur fournit un input, que ce soit une recherche par mot-clé ou un autre type de requête, la fonction query() analyse cet input et génère une requête SQL correspondante. La requête SQL est ensuite exécutée et retourne une liste de résultats. Dans ce cas précis, cette liste contient tous les liens qui correspondent aux critères de recherche de l'utilisateur. Une fois que nous avons récupéré cette liste de liens, nous passons à l'étape suivante qui consiste à intégrer ces liens dans notre page HTML. Pour ce faire, nous parcourons chaque lien de la liste et l'ajoutons à l'emplacement approprié sur la page. Ces emplacements sont prédéfinis et structurés de manière à accueillir les liens de manière ordonnée et esthétique. Cette insertion permet d'offrir une expérience utilisateur fluide et intuitive, permettant à l'utilisateur de voir et d'accéder facilement aux liens pertinents.

Ensuite, nous avons implémenté la requête qui permet de récupérer tout les mots-clés de la base de données, à l'aide d'une fonction. Chaque élément récupéré est sous un format illisible et doit être transformé, on en extrait ainsi chaque mot. A partir de là, nous pouvons créer une structure pour calculer la pondération de chaque mots, et enfin, un par un, les ajouter au dictionnaire. Si le mot existe ( "cat" par exemple ), la pondération du mot dans le dictionnaire est mise à jour, sinon, le mot est créé ( exemple : google ). Ainsi, à chaque chargement de dictionnaire vers premium ( changement de langue, ouverture du

programme ), les mots sont ajoutés au dictionnaire. La requête n'est faite que une fois, au démarrage de la première page web.

Nous avons ajouté une nouvelle fonctionnalité qui permet aux utilisateurs de rester sur notre page pour continuer leurs recherches sans avoir à ouvrir une nouvelle fenêtre ou un nouvel onglet. Cette amélioration évite l'accumulation de multiples pages ouvertes, ce qui pourrait devenir rapidement ingérable et encombrer le navigateur de l'utilisateur. Grâce à cette fonctionnalité, l'expérience de navigation devient plus fluide et agréable, permettant aux utilisateurs de trouver rapidement et facilement les informations qu'ils recherchent sans la distraction des nombreuses fenêtres ouvertes.

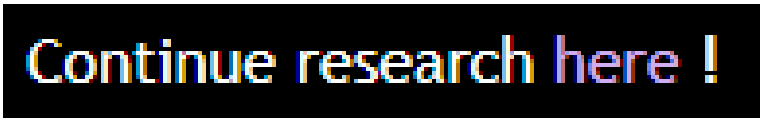



FIGURE 32 – New Tab

# 8 Avancement & répartition

| Répartitions | | | | | |
|---|---|---|---|---|---|
| | Léo | Justine | Quentin | Alexandre | Avancement |
| Crawler | Scalable Bloom Filter | | | | 100% |
| | Seeding + Expanding | | | | 100% |
| | Performance optimization | | | | 100% |
| Scraper | | Data Extraction | | | 100% |
| | | Data Cleansing | | | 100% |
| | | Inverted Index | | | 100% |
| User Input | | | Parsing | | 100% |
| | | | Lexing | | 100% |
| | | | SQL Queries | | 100% |
| | | | | Spell Checker | 100% |
| | | | | Suggesting | 100% |
| | | | | Keyword Filtering | 100% |
| Visualizer | | | Interface | Interface | 100% |

- $1^{ère}$ soutenance
- $2^{ème}$ soutenance
- $3^{ème}$ soutenance

# 9 Ressentis Individuels

## 9.1 Quentin Couavoux

Pour moi, ce projet a été l'un des meilleurs depuis le début de mon cursus à EPITA. Tout d'abord, en raison de la liberté de choix de projet offerte à chaque groupe, nous avons pu choisir ce que nous voulions réaliser, ce qui a donné lieu à des projets très diversifiés pour chaque groupe avec aucune ressemblance. Cette flexibilité a permis à chaque groupe de s'engager dans un domaine qui les passionne, favorisant ainsi la motivation et l'investissement personnel dans le travail. Le groupe avec lequel j'ai travaillé a été exemplaire. Nous avons formé une équipe efficace, où chacun comprenait les attentes des autres et faisait preuve d'altruisme. La collaboration a été fluide, avec une excellente communication et un soutien mutuel constant. Chaque membre a apporté ses compétences uniques, enrichissant ainsi notre projet commun. Un des avantages majeurs de ce projet était le fait que nous n'étions pas limités par les ordinateurs de l'école. Cela nous a permis d'utiliser une large variété de bibliothèques et de technologies, nous donnant ainsi la liberté d'explorer des outils modernes et adaptés à nos besoins spécifiques. Toutefois, cette liberté a également introduit des défis. Certaines bibliothèques que nous souhaitions utiliser étaient peu, voire pas du tout, documentées. Cette absence de documentation claire a parfois engendré des moments de confusion et a nécessité des efforts supplémentaires pour comprendre et intégrer ces outils dans notre projet. Malgré les difficultés rencontrées, telles que la gestion de bibliothèques peu documentées et les obstacles techniques imprévus, j'ai passé un excellent moment à travailler sur ce projet. Certains moments ont été plus difficiles que d'autres, mais ces défis ont été l'occasion de développer notre résilience et notre capacité à résoudre des problèmes complexes. Si je devais refaire ce projet, je le referais avec plaisir. Ce projet m'a apporté de nombreuses connaissances techniques et m'a permis de développer de nouvelles compétences en programmation et en gestion de projet. De plus, j'ai acquis une meilleure compréhension du travail en équipe et de l'importance de la communication et de la coopération. Donc ce projet a été une expérience extrêmement enrichissante sur de nombreux plans. Je suis donc reconnaissant pour cette opportunité et j'attends avec impatience les projets futurs qui me permettront de continuer à apprendre et à grandir.

## 9.2 Alexandre Colliard

Tout d'abord, la machine virtuelle ubuntu que j'utilise n'a pas aimé l'installation d'une librairie en c au début de l'année et est devenue complètement inutilisable à cause d'un blue screen qui est apparu lors de l'exécution de commandes administrateur pour régler le problème. Je me suis retrouvé à faire tout mes tp/projets sur les pc de l'école, et c'est lors du projet s4 que c'est devenu problématique. En ajoutant la librairie rocket ( même chose pour toutes les autres librairies d'interface ), le temps de compilation est passé de 1min à 1h20 sur les pc de l'école. N'ayant aucune autre possibilité, j'ai pris le projet bien en avance et j'ai passé des journées entières à coder à Epita à coup de 3 compilations par jours. J'ai essayé d'autres solutions comme coder sur d'autres pc et de le faire compiler par qqn d'autres ( pushs sous le nom de averageDoomFan ). Mais aucune n'a été vraiment efficace. Arrivant à la fin de projet, j'ai pu terminer toutes les fonctionnalités dans les temps et j'ai pu faire le merge de ma partie avec quentin. Nous avons travaillé à 2 sur son pc ( via discord ) et je l'ai guidé lors du développement des quelques fonctionnalités qui nécessitaient ma partie. Tout les deux nous avons travaillé très efficacement sur la dernière semaine de projet ( sur le merge de nos parties, fixs, améliorations pratiques et visuelles en tout genre ). Malgré cela, j'ai passé de très bons moments à travailler sur ce projet, me documenter et coder ( malgré les temps de compilations démoniaques ). J'ai énormément appris, que ce soit sur le sujet ( dictionnaires, moteurs de recherches et les parties de mes camarades ) et sur la manière d'optimiser les structures de données. La satisfaction de l'utilisation de notre moteur de recherche est incomparable, je ne suis que fier du travail accompli. J'ai trouvé que ce groupe était génial, tout le monde a su travailler sérieusement tout en gardant un bon sens de l'humour et de la positivité. C'est la première fois que tout se passe correctement dans un projet de groupe, et c'est sûrement grâce aux liens que nous avons pu créer.

## 9.3 Léo Menaldo

De mon côté, je ressors vraiment grandi d'une telle expérience, elle me faisait rêver et j'ai vraiment pu accomplir ce que j'espérais, je suis fier de mon groupe pour avoir mené à bien ce projet. J'ai pu découvrir énormément de choses autour d'internet, que ce soit dans la façon dont on fait des requêtes, la manière de les traiter, ou même le simple fait d'avoir travaillé autour de ce sujet pendant plusieurs mois. Cette idée "d'explorer" le web m'enivrait profondément. De plus, le langage imposé, étant un peu coriace la plupart du temps, rajoutait une difficulté supplémentaire qui fût plaisante à surmonter.
Si je devais refaire un projet d'une telle envergure, ce serait sûrement avec le même groupe. Je me sentais vraiment bien dans celui-ci et j'y ressentais une très forte alchimie. Malgré tout cela, j'ai ressenti personnellement comme un manque de tâches à accomplir, jusqu'à lors, pour chacun des projets auxquels j'ai participé, je voulais toujours m'occuper des tâches les plus ardues et celles qui étaient pour moi les plus importantes à réaliser. Mais c'est grâce à DroXyd que j'ai compris la puissance d'un groupe soudé, tout était bien harmonisé et homogène, j'ai eu l'impression que nous avons pu tous nous amuser sur nos parties respectives. En réalité, je pense que ce projet a été un des meilleurs auquel j'ai contribué jusqu'à lors...

## 9.4 Justine Gueulet

J'ai beaucoup apprécié travailler sur ce projet car non seulement j'ai énormément appris sur tout ce qui est navigateur de recherche, mais l'ambiance du groupe a beaucoup aidé. Nous nous étions embarqué dans un gros projet qui nous semblait presque irréalisable au début, mais avec l'entraide et la bienveillance qui était au sein de notre groupe, nous avons su bien avancer sans jamais avoir de retard. Même quand on avait des soucis avec nos parties, on s'appelait et on réglait tout cela ensemble en s'amusant en même temps.
Je suis vraiment contente que tout le monde ait aidé sur ce projet, que tout le monde ait mis du sien et qu'on a tous complété nos parties, car dans certains projets passés cela ne s'était jamais produit.
C'était un très beau projet, sans doute le meilleur pour ma part, et je suis déçe de devoir partir de Lyon l'année prochaine car j'aurais voulu refaire partie du même groupe pours les projet d'ing.

# 10 Ressenti Global

Notre ressenti global est très positif, grâce à l'ambiance mais aussi au projet que nous avons réussi à finaliser. Ce projet, même s'il reste complexe, est aussi passionnant qu'intéressant.

De plus, nous avons amélioré la rapidité et l'optimisation de certains de nos algorithmes comme nous l'esperions depuis la deuxième soutenance, pour que ces algorithmes soient non seulement plus efficaces mais qu'ils puissent nous donner une précision satisfaisante.

Nous avons réussi à implémenter quelques tâches en plus que nous expliquerons en détail lors de notre soutenance finale. Nous somme rester confiants tout le long du projet, et nous avons hâte de présenter son résultat final.

# 11 Conclusion

D'un point de vue global, notre projet a très bien avancé depuis la première et la deuxième soutenance qui s'étaient déjà très bien passées. Encore une fois, nous avons tous réussi à finir nos tâches tout en gardant une bonne ambiance au sein du groupe. Malgré quelques problèmes rencontrés, par exemple avec la base de données, nous avons su y faire face et même si l'implémentation de la base de données n'est pas aussi optimisées que ce qu'on espérait, on a réussi à tout faire marcher. C'était un très beau projet où on a mis beaucoup d'effort, et nous sommes fières de notre projet et de ses progrès.

# 12 Annexes

## 12.1 User Input Processing

https://diesel.rs/guides/getting-started

https://rocket.rs

https://www.youtube.com/watch?v=_pPtsPNdNrc

https://www.youtube.com/watch?v=xzWwpXx2olc

https://docs.diesel.rs/master/diesel/deserialize/trait.FromSql.htmltymethod.from_sql

https://docs.diesel.rs/2.0.x/diesel/query_builder/struct.SelectStatement.html

https://rust-exercises.com/07_threads/02_static

https://dhghomon.github.io/easyrust/Chapter40.html

## 12.2 Crawler

https://systemdesign.one/bloom-filters-explained/
https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/
https://dergipark.org.tr/en/download/article-file/
https://gsd.di.uminho.pt/members/cbm/ps/dbloom.pdf
https://dridk.me/bloom-filter.html
https://blog.octo.com/le-filtre-de-bloom
https://fr.wikipedia.org/wiki/SHA-256
https://fr.wikipedia.org/wiki/SMD5 https://fr.wikipedia.org/wiki/Hypertext-Transfer-Protocol
https://www.techtarget.com/whatis/definition/multithreading
https://crates.io/crates/reqwest

Liens de semencement du crawler :
https://web.archive.org/web/20170131181820/http://www.dmoz.org/
https://en.wikipedia.org/wiki/42
https://www.lepoint.fr/eureka/qu-est-ce-qu-une-eclipse-solaire-tota
https://www.psychologue.net/articles/largent-fait-il-le-bonheur
https://www.lequipe.fr/Football/Coupe-du-monde/
https://it.wikipedia.org/wiki/Pizza
https://olympics.com/
https://www.nytimes.com/
https://de.wikipedia.org/wiki/Attack_on_Titan
https://es.wikipedia.org/wiki/Pulp_Fiction

## 12.3 Scraper

### Data Extraction

https://www.w3schools.com/tags/tagtitle.asp

https://www.view-page-source.com/

https://stackoverflow.com/questions/22461463/how-can-i-tell-if-the-page-source-is-xhtml-html-or-xml

https://html.com/tags/p/: :text=The-end-of-the-paragraph,a

https://developer.mozilla.org/fr/docs/Web/HTML/Element/p

### Scraper

https://users.rust-lang.org/t/how-to-check-for-internet-connection/89893/5

https://hyper.rs/

https://stackoverflow.com/questions/150750/hashset-vs-list-performance

https://stackoverflow.com/questions/.

https://users.rust-lang.org/t/in-memory-btreemap-like-crate-that-supports-acid-transactions/106457/2

https://www.dotnetperls.com/btreemap-rust

https://iq.opengenus.org/hashmap-and-btreemap-rust/

https://www.reddit.com/btreemap-v-hashmap

https://youtu.be/vZAXpvHhQow?si=6QycA6u2b3DBkSQy

https://youtu.be/OymqCnh-APA?si=IHNMOUhmKGko3x3E

https://youtu.be/D2V1okCEsiE?si=N-VTsyA2fgM5Zmpa

https://programminghistorian.org/analyzing-documents-with-tfidf

https://users.rust-lang.org/t/watch-out-for-nans/70016/5

https://datascientest.com/tf-idf-intelligence-artificielle

https://docs.rs/reqwest/latest/reqwest/

# 13 Visualisation de l'historique de développement

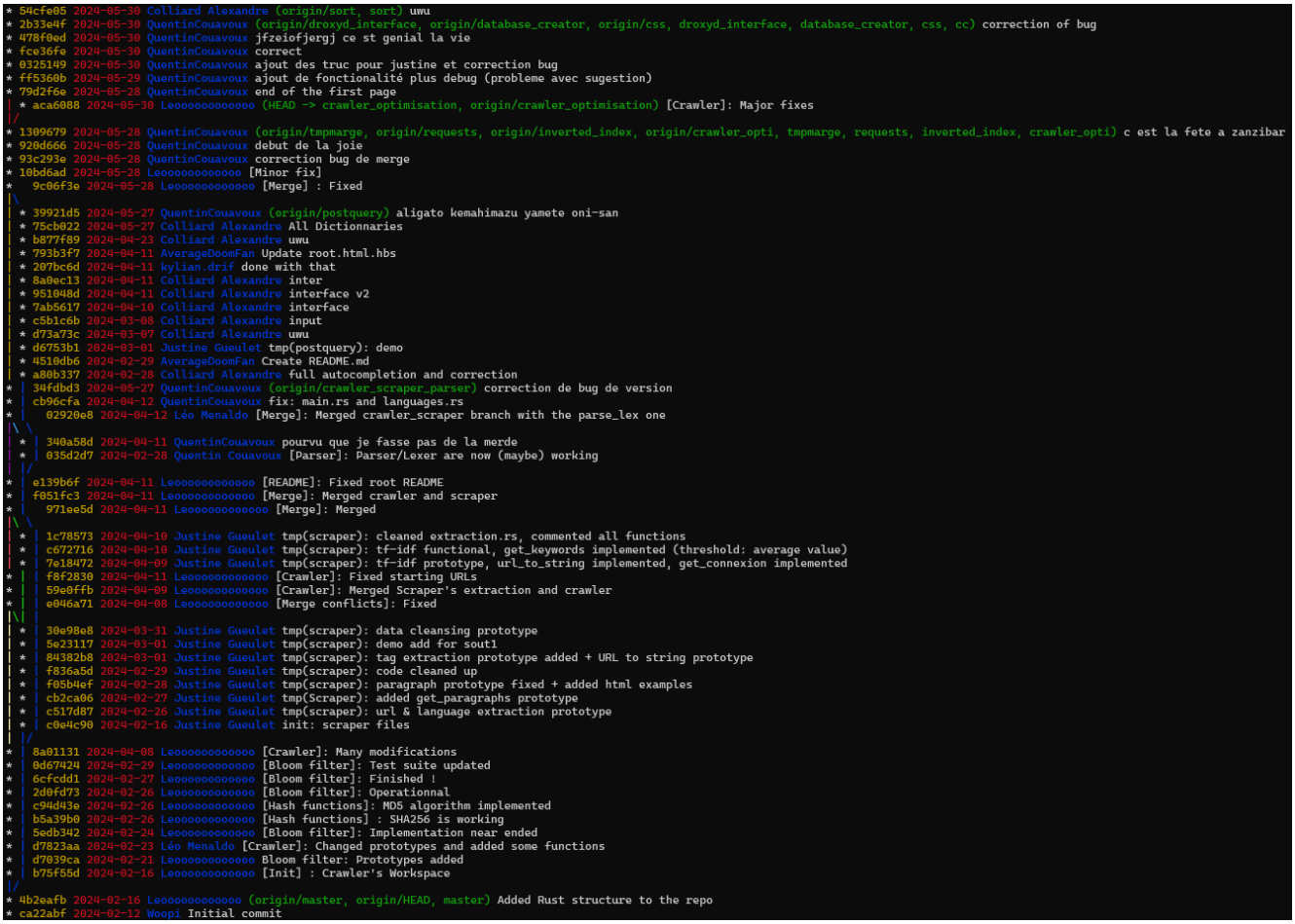



FIGURE 33 – Commit