Lucene

Semifir

Enzo Radnaï / Adrien Vossough

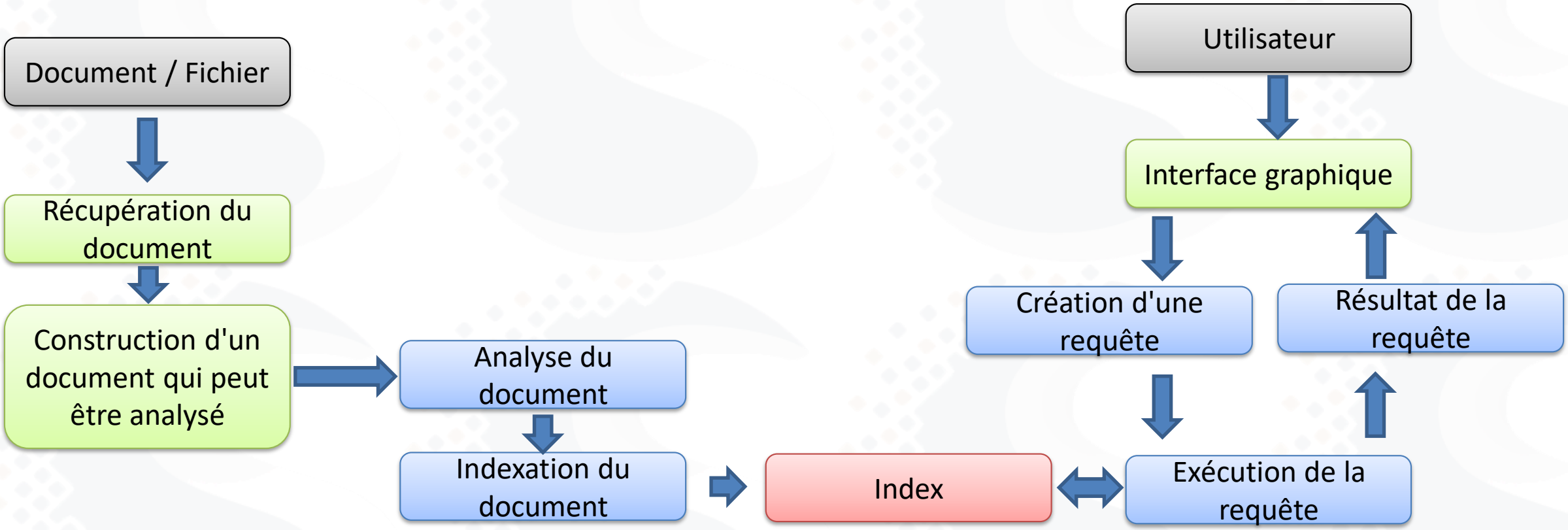
https://www.semifir.com/

Qu'est-ce qu'Apache Lucene ?

- Est un moteur de recherche de texte écrit en Java
- Ce n'est pas une application Web mais une bibliothèque avec une API
  - Classes d'analyse et découpage de texte
  - Constructeurs de requêtes de recherche
  - Constructeurs de documents à indexer
  - Gestionnaires d'index
- Il permet d'ajouter la capacité de recherche à une application
- Lucene est indépendant de tous les formats de fichiers ou des bases de données tant que le texte peut être extrait
- Propose l'indexation incrémentale plutôt qu' une indexation par lots : un index peut être mis à jour, des entrées individuelles peuvent être ajoutées ou supprimées.

Lucene : fonctionnement

- Lucene est indépendant de tous les formats de fichiers tant que le texte peut être extrait



Lucene : exemple de document à traiter

Les informations proviennent de fichiers ou flux et sont **transformés en documents** sur lesquels Lucene peut travailler.

Exemple :


```
{
  "livres": [
    {
      "titre": "Eloquent JavaScript, Second Edition",
      "auteur": "Marijn Haverbeke",
      "published": "2014-12-14T00:00:00.000Z",
      "editeur": "No Starch Press",
      "pages": 472,
      "description": "JavaScript lies at the heart of almost every modern web application, from social apps to the newest browser-based games. Though simple for beginners to pick up and play with, JavaScript is a flexible, complex language that you can use to build full-scale applications."
    },
    {
      "titre": "Learning JavaScript Design Patterns",
      "auteur": "Addy Osmani",
      "published": "2012-07-01T00:00:00.000Z",
      "editeur": "O'Reilly Media",
      "pages": 254,
      "description": "With Learning JavaScript Design Patterns, you'll learn how to write beautiful, structured, and maintainable JavaScript by applying classical and modern design patterns to the language. If you want to keep your code efficient, more manageable, and up-to-date with the latest best practices, this book is for you.",
    }
  ]
}
```


Lors de la création de l'index, il est possible de sélectionner les métadonnées à inclure.

Lucene

Les documents sont ensuite tokenisés, segmentés en mots.
Pour éviter de "casser" les mots composés, il est possible d'inclure des dictionnaires à Lucene.

Lucene effectue également une normalisation, par exemple : les lettres majuscules passent en minuscules.

Un tri des mots est fait via la mesure TF-IDF (term frequency-inverse document frequency) . obtenir les résultats les plus pertinents ou les plus récents, les algorithmes du moteur de recherche Lucene rendent cela possible.

Les documents sont indexés, les utilisateurs peuvent faire des requêtes :
https://lucene.apache.org/core/8_0_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package.description

Lucene : indexation

Lors de l'analyse du document

- il est possible de retirer les mots trop communs avec un dictionnaire
- Retrait des signes de ponctuation
- Passage en minuscule

Texte original

JavaScript lies at the heart of almost every modern web application, from social apps to the newest browser-based games. Though simple for beginners to pick up and play with, JavaScript is a flexible, complex language that you can use to build full-scale applications.

Texte après analyse

javascript lies heart almost every modern web application social apps newest browser-based games simple beginners play javascript flexible complex language build full-scale applications

Lucene : Scoring

Lorsque le document est indexé, il peut être recherché.

La recherche est influencée par la pertinence/scoring.

Les facteurs pris en compte :

- **tf = term frequency :** fréquence du terme dans un document
- **idf = inverse document frequency** : mesure le nombre de fois ou le terme apparaît dans les différents documents. Un terme trop fréquent perd en pertinence
- **coord** : nombre de termes dans la requête retrouvé dans le document
- **lengthNorm** : mesure l'importance d'un terme d'après le nombre total présent dans un champs précis du document
- **queryNorm** : normalisation de la requête pour la comparer aux résultats de précédentes requêtes
- **boost (index) :** possibilité d'augmenter la pertinence des termes présent dans un champs précis du document lors de l'indexation
- **boost (query) :** possibilité d'augmenter la pertinence des termes présent dans un champs précis du document lors de la recherche

Lucene : classes pour indexer les données

IndexWriter
Classe qui donne accès aux index en écriture (création, ajout de document, optimisation, …)

Analyzer
Classes permettant le découpage du texte en « token » (mot) et la normalisation du texte à indexer.

- **SimpleAnalyzer,** découpe le texte en mots minuscules.
- **StopAnalyzer,** découpe le texte en mots minuscules et supprime les mots vides (le, la, de …)
- **StandardAnalyzer,** combine les deux analyzer précédents

Document
Unité élémentaire d'information. Par exemple, indexer tous les fichiers Word d'un répertoire va ajouter dans l'index un Document Lucene par fichier. Ce sont des Documents qui sont retournés dans la liste de résultats d'une recherche. Un document est constitué de champs « Field » (nom / valeurs).

Field
Il s'agit d'un sous élément d'un document, tel que : titre, auteur, date de publication, url et le texte du fichier Word, PDF ou HTML.

Lucene : classes pour rechercher les données

IndexSearcher
Classe qui donne accès aux index en recherche

Analyzer
Tout comme pour l'indexation les analyzer font partie du processus de recherche fin de normaliser les critères de recherche :

QueryParser
parser de requête

Query
un objet qui représente la requête de l'utilisateur et utilisé par un IndexSearcher.

Hits
Une collection d'éléments résultats de la recherche

Hit
Un élément de la collection des résultats

Document
Un document retrouvé

Lucene: Motifs de recherche

Single Term : Un mot. Contrairement à des moteurs de recherche célèbres, Lucene suppose que vous savez comment le terme est correctement écrit. Une faute d'orthographe et vous obtiendrez un résultat négatif.
Exemple : voiture

Phrase : séquences de mots. Les termes et leur l'ordre sont déterminants pour le résultat
Exemple : "Adrien développe en Java".

Wildcard Searches : Il remplacent un ou plusieurs caractères dans la requête de recherche. Les caractères génériques peuvent être utilisés à la fin et au milieu d'un terme, mais pas au début.

- **?** : Le point d'interrogation remplace **un** caractère. Exemple : Adr?en
- ***** : Représente 0 ou plusieurs caractères. Exemple : Auto*

Regular Expression Searches : Des expressions régulières pour rechercher plusieurs termes.
Exemple : /[MT]on/

Fuzzy Searches : Recherche vague qui permet d'avoir une tolérance aux fautes dans un mot. La formule Damerau-Levenshtein évalue les similitudes. Des distances de 0 à 2 sont autorisées.
Exemple : Adrien~1

Proximity Searches : Permet une approximation pour les phrases. Par exemple, qu'entre 2 termes, il peut y avoir jusqu'à 5 mots . Exemple : "Adrien v"~5

Lucene: Motifs de recherche

Range Searches : Permet une recherche bornée. Utilisez TO pour délimiter les deux termes.
Les crochets indiquent une inclusion tandis que les accolades une exclusion.
Exemple : [Adrien TO Enzo] ou {100 TO 500} ou [500 to 1000 }

Boosting : Permet d'amplifier la pertinence d'un term avec le circonflexe suivie d'une valeur.
Exemple : Auto^2 rouge

Boolean Operators :

- **AND** : La présence de tous les termes sont obligatoires. Exemple : Voiture && rouge
- **OR** : Recherche par défaut. Un des termes doit être présent. Exemple : Voiture rouge
- **+** : Rend un terme obligatoire. Exemple : +Voiture rouge
- **NOT** : Exclusion d'un terme avec le signe – ou !
  Exemple : Voiture rouge –bleu

Grouping : Les parenthèses permette de grouper un motif. exemple : Voiture ET (rouge OU bleu).

Escaping Special Characters : Il faut parfois échapper un caractère pour que celui-ci ne soit pas pris pour un opérateur : "Où est ma voiture\?"