Marcel Claramunt, Biel Escorsell, Xavier García, Jan Samaranch

Projectes de Programació

Macrosoft Word

Marcel Claramunt, Biel Escorsell, Xavier García, Jan Samaranch

Abstracte

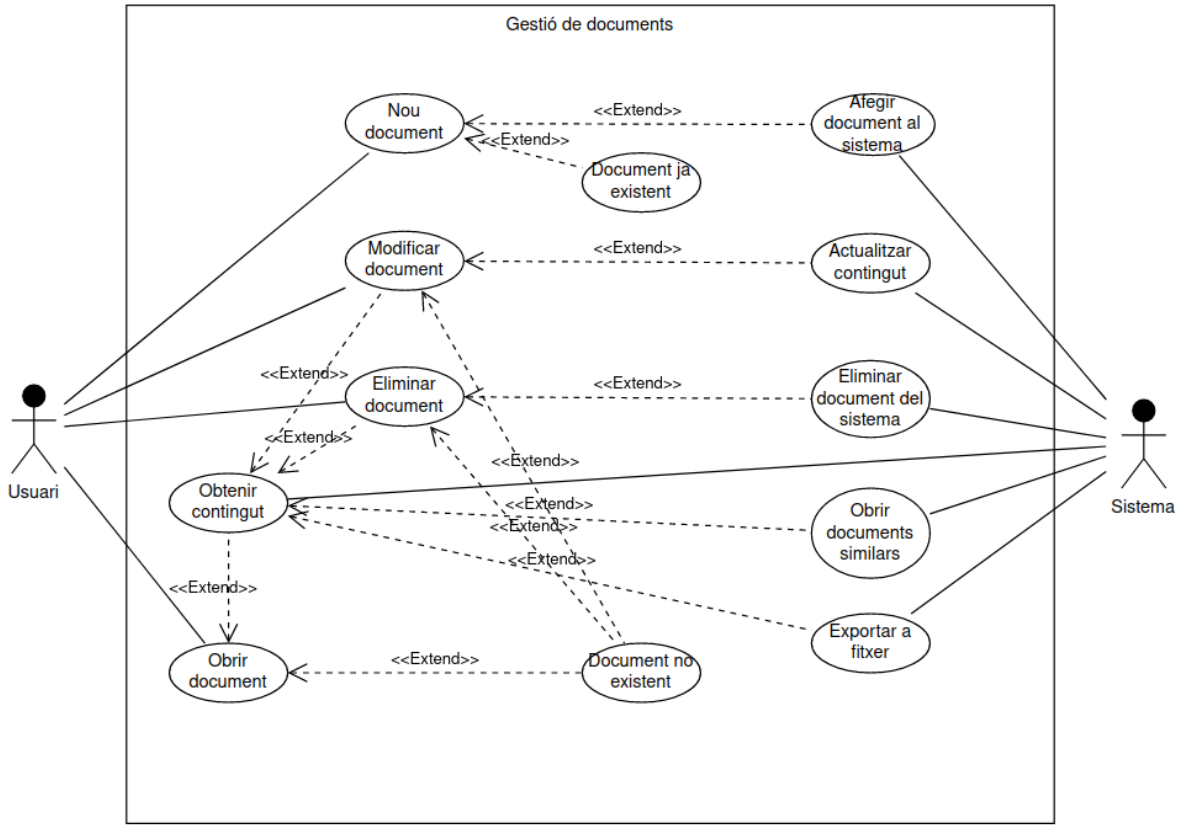
Índex

Marcel Claramunt, Biel Escorsell, Xavier García, Jan Samaranch

1. Introducció

2. Casos d'Ús

2.1. Gestió de Documents



Imatge 1. Diagrama de Casos d'Ús – Gestió de Documents

UC 1 – Nou Document

L'usuari indica el nom de l'autor, el títol del document i el seu contingut. Si no hi ha cap document al sistema amb els mateixos autor i títol, llavors s'afegeix al sistema un nou document amb aquest autor, títol i contingut.

UC2 – Eliminar Document

L'usuari indica un nom d'autor i un títol. Si existeix al sistema un document identificat per aquest autor i títol, s'elimina del sistema.

UC3 – Modificar Contingut

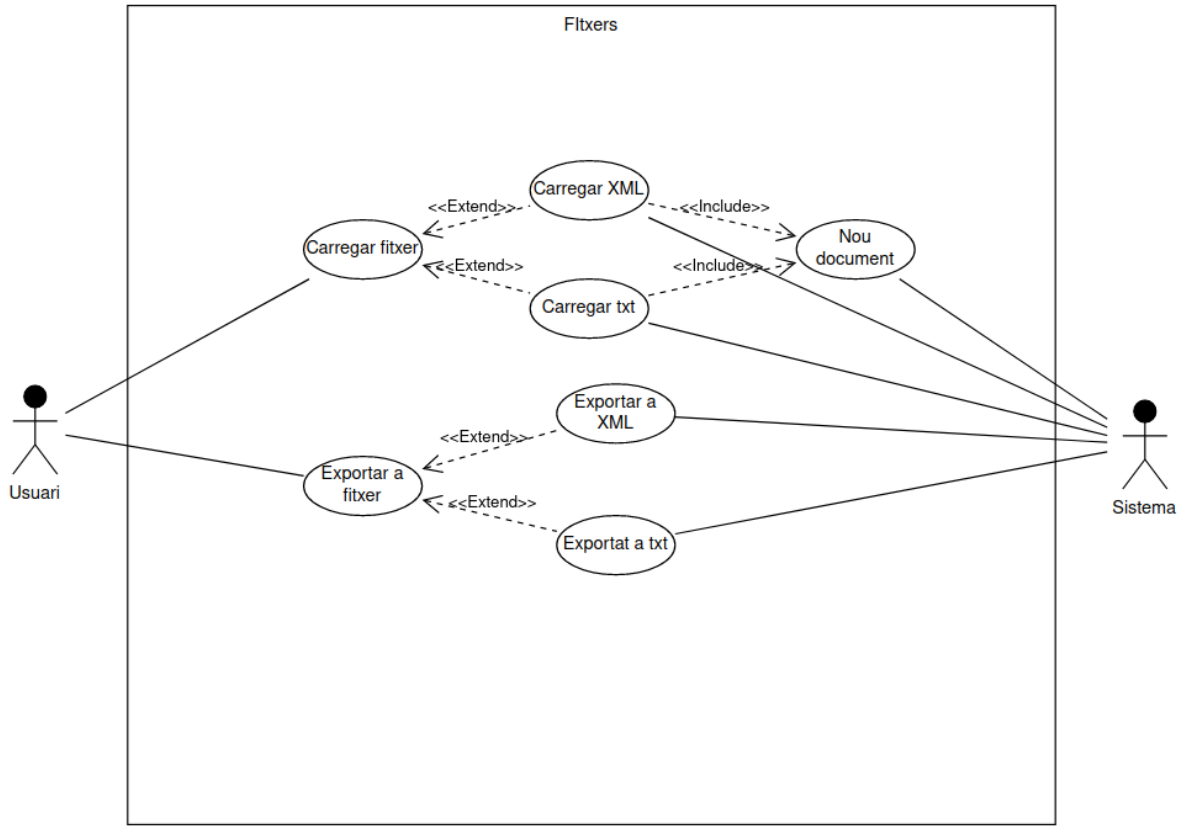
L'usuari indica un nom d'autor i un títol. Si existeix al sistema un document identificat per aquest autor i títol, es demana el nou contingut per al document i es sobreescriu el que hi havia.

Marcel Claramunt, Biel Escorsell, Xavier García, Jan Samaranch

UC4 - Obrir Document

L'usuari indica un nom d'autor i un títol. Si existeix al sistema un document identificat per aquest autor i aquest títol, es retorna el seu contingut. Un cop obert un document, l'usuari pot modificar el seu contingut, eliminar el document, cercar documents amb contingut similar o exportar a un fitxer.

2.2. Gestió de Fitxers



Imatge 2. Diagrama de Casos d'Ús – Gestió de Fitxers

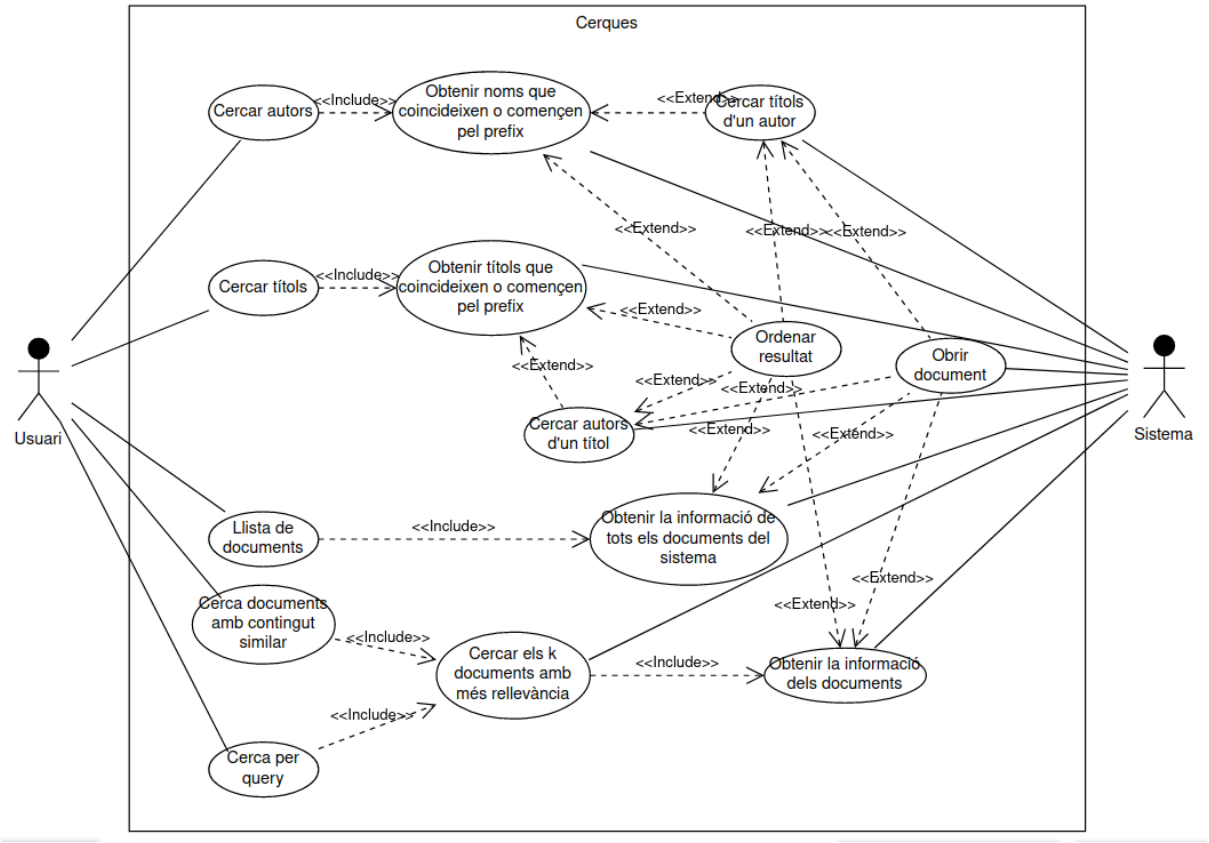
UC5 – Carregar Fitxer

L'usuari indica un fitxer del seu sistema operatiu que conté nom de l'autor, títol i contingut i es crea un nou document amb aquesta informació si al sistema no hi ha cap document que s'identifiqui amb aquest títol i autor.

UC6 - Exportar a Fitxer

L'usuari indica el nom d'un autor, un títol i un format. Si al sistema existeix un document amb aquest autor i títol, s'exporta el seu contingut i informació a un fitxer amb el format escollit.

2.3. Cerques



Imatge 3. Diagrama de Casos d'Ús – Cerques

UC7 – Cerca d'Autors

L'usuari introdueix una seqüència de caràcters, que poden ser tant un nom complet d'autor, un prefix o una seqüència buida. Es retornen tots els noms dels autors que coincideixen o comencen per la seqüència. Un cop obtinguts, l'usuari pot ordenar el resultat de forma ascendent i descendent, i obtenir, indicant un dels noms, un llistat dels títols de tots els documents que té l'autor en el sistema. L'usuari pot ordenar aquests nous resultats o, escollint un dels títols, obrir el document identificat pel títol i l'autor.

UC8 – Cerca de Títols

L'usuari introdueix una seqüència de caràcters, que poden ser tant un títol complet, un prefix o una seqüència buida. Es retornen tots els títols que coincideixen o comencen per la seqüència. Un cop obtinguts, l'usuari pot ordenar el resultat de forma ascendent i descendent, i obtenir, indicant un dels títols, un llistat dels noms dels autors de tots els documents en el sistema que tenen aquest títol. L'usuari pot ordenar aquests nous resultats o, escollint un dels noms, obrir el document identificat pel títol i aquest autor.

UC9 – Llista de Documents

Es retorna la informació (títol, nom de l'autor, data de creació i última data de modificació) de tots els documents que hi ha al sistema. L'usuari pot ordenar el resultat de forma ascendent i descendent per cada camp de la informació retornada, i indicant un dels documents, obrir-lo per veure el seu contingut.

UC10 – Cerca de Documents Similars

L'usuari indica un nom d'autor, un títol i, opcionalment, un enter *k* (sinó s'utilitzarà un valor per defecte). Si existeix al sistema un document identificat per aquest autor i títol, es retorna la informació (títol, nom de l'autor, data de creació i última data de modificació) dels *k* documents amb contingut més semblant al document identificat pel nom de l'autor i títol indicats. Els resultats obtinguts es poden ordenar de forma ascendent i descendent per cada camp de la informació retornada i, indicant un dels documents, es pot obrir per veure el seu contingut.

UC11 – Cerca per query

L'usuari introdueix un conjunt de paraules i, opcionalment, un enter *k* (sinó s'utilitzarà un valor per defecte). Es retorna la informació dels documents més rellevants, pel que fa al contingut, per aquest conjunt de paraules. Un cop obtinguts els resultats es poden ordenar o obrir un dels documents.

2.4. Gestió d'Expressions Booleanes

UC12 – Afegir expressió booleana

L'usuari introdueix un nom i una expressió booleana formada pels operadors &, | i !, conjunts de paraules (delimitats per {}), seqüències de paraules (delimitades per "") o paraules soltes com a operands. Si l'expressió és correcte i no hi ha cap altra expressió al sistema amb el mateix nom, s'afegeixen el nom i l'expressió al sistema.

UC13 – Modificar expressió booleana

L'usuari indica el nom de l'expressió booleana a modificar i la nova expressió per la qual es vol canviar. Si existeix al sistema una expressió identificada per aquest nom i la nova expressió és correcte, l'expressió identificada pel nom indicat passa a ser la nova expressió.

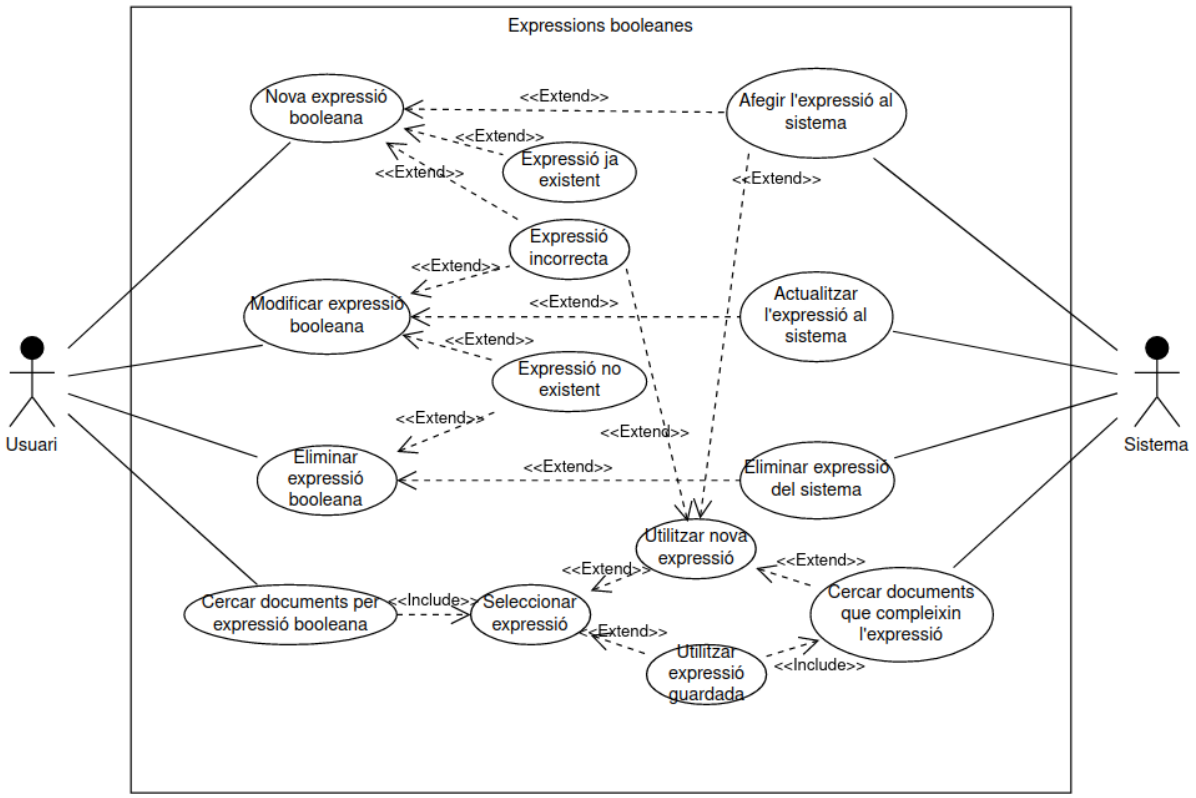
UC14 – Eliminar expressió booleana

L'usuari indica el nom d'una de les expressions booleanes guardades al sistema. Si existeix al sistema una expressió identificada per aquest nom, s'elimina.

UC15 – Cerca per expressió booleana

L'usuari escull si vol utilitzar una de les expressions booleanes guardades al sistema o una nova. Si escull l'opció d'utilitzar una de les guardades, es mostren les expressions guardades i l'usuari n'ha de seleccionar una. Si decideix utilitzar una nova, la introdueix manualment i es confirma que és una

expressió correcte. Un cop escollida una expressió es retorna la informació de tots els documents del sistema que contenen una frase que satisfà l'expressió. Obtinguts els resultats, l'usuari pot decidir ordenar els resultats o obrir un dels documents.

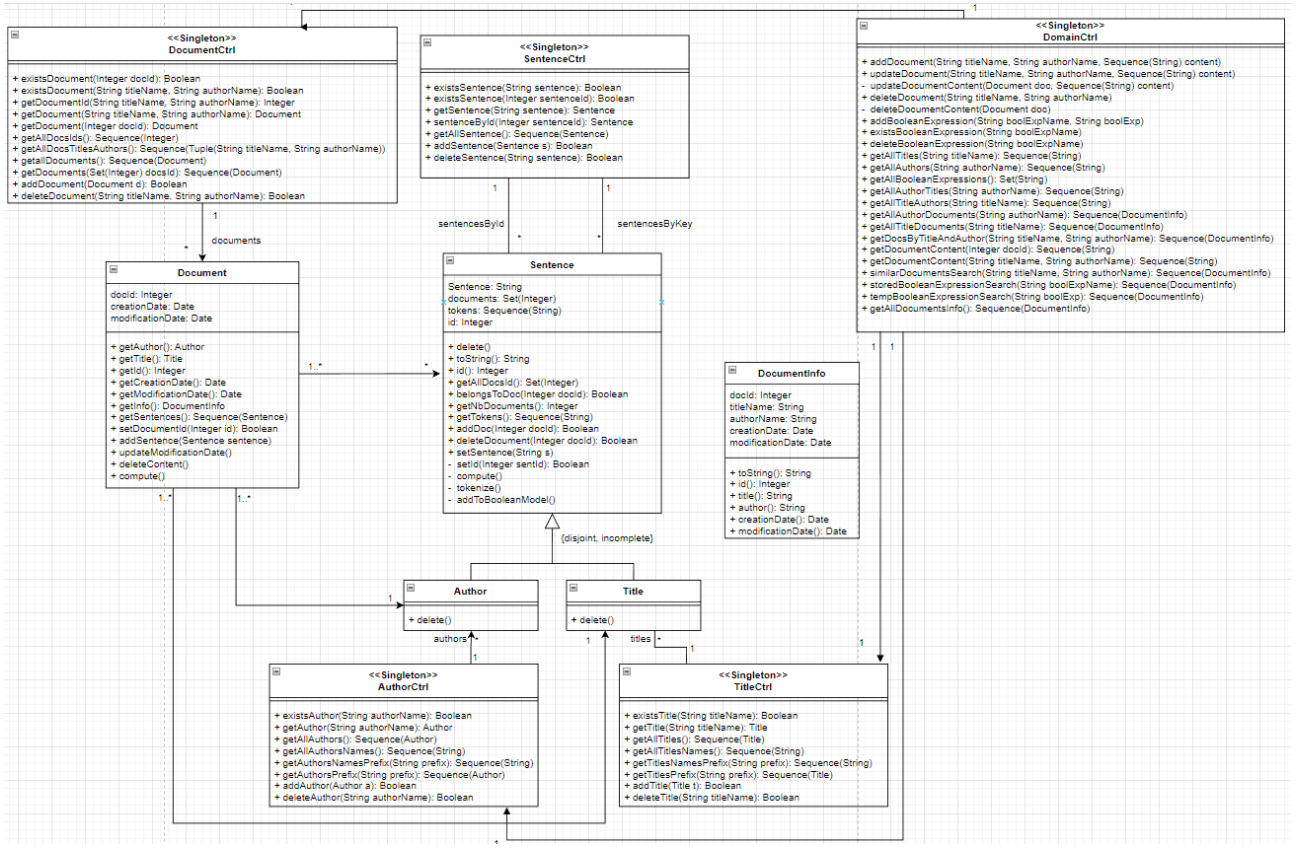


Imatge 4. Diagrama de Casos d'Ús – Gestió d'Expressions Booleanes

Marcel Claramunt, Biel Escorsell, Xavier García, Jan Samaranch

3. Model Conceptual

3.1. Diagrama UML



Imatge 5. Diagrama de Classes del Domini

Restriccions textuals:
RT1 - Claus externes, Sentence(Sentence), Document(title + author), DocumentInfo(docId), BooleanExpression(expression), Index(invertedIndex + directIndex)

3.2. Explicació de les Classes

Document

Cada objecte pertanyen a aquesta classe representa un Document en el nostre Sistema, Aquest conte un element de la classe Author, un de la classe Title i un ArrayList de elements de la classe Sentence.

Document Controller
Classe de tipus singleton que guarda i gestiona les instàncies de Document que hi ha al sistema.

Sentence
Cada instància d'aquesta classe representa una frase. Conté un ID que identifica la frase, un set d'enters que representa els documents on apareix la frase i un ArrayList de les paraules de la frase filtrades.

Sentence Controller
Classe de tipus singleton que guarda i gestiona les instàncies de Sentence que hi ha al sistema. Les instàncies de Sentence que hi ha són només aquelles que apareixen, com a mínim, en un document.

Title
Cada instància d'aquesta classe representa un títol d'un document. Hereta de Sentence i en aquesta classe, el set d'enters que a Sentence indiquen a quins documents apareix la frase, a Title indiquen quins documents tenen aquest títol.

Title Controller
Classe de tipus singleton que guarda i gestiona les instàncies de Title que hi ha al sistema. Les instàncies de Title que hi ha són només aquelles que estan relacionades, com a mínim, amb un document.

Author
Cada instància d'aquesta classe representa un autor d'un document. Hereta de Sentence i en aquesta classe, el set d'enters que a Sentence indiquen a quins documents apareix la frase, a Author indiquen quins documents tenen aquest autor.

Author Controller
Classe de tipus singleton que guarda i gestiona les instàncies d'Author que hi ha al sistema. Les instàncies de Author que hi ha són només aquelles que estan relacionades, com a mínim, amb un document.

4. Indexació – Estructures de Dades

4.1. Justificació

Una de les principals funcionalitats de l'aplicació és la de realitzar *queries* sobre els documents que hi hem afegit. D'una banda, s'ha de poder cercar per *similaritat*, la qual cosa és, a priori, difícil de decidir. Farem servir un dels models més utilitzats dins el món de l'*Information Retrieval*, el *model vectorial* amb pesos *tf-idf*.

D'altra banda, també volem cercar frases – no documents – que compleixin certes expressions booleanes. Per a aquest fi, utilitzarem una modificació del model vectorial. A part d'indexar per frases enlloc de per documents, guardarem les posicions en què apareix cada terme – així, podrem cercar expressions que continguin una seqüència de paraules (on l'ordre és rellevant).

4.2. Model Vectorial – Fitxer Directe

El *model vectorial* és una abstracció que representa cada document $d \in D$ com un *vector de freqüències* de cada paraula (*terme*) d'un *vocabulari* T. D és la col·lecció dels documents que considerem, i T és el conjunt de termes que apareixen en algun dels documents de D.

Per exemple, si tenim els documents d_2 = "el noi és alt" i d_2 = "el joc és el seu", la seva representació vectorial (normalment anomenada *fitxer o índex directe*) seria la següent:

Document	el	noi	és	alt	joc	seu
1	1	1	1	1	0	0
2	2	0	1	0	1	1

Ara bé, a la pràctica, enlloc de tenir un vector de $|T|$ posicions, guardem només aquells termes amb freqüència no nul·la. A més, per tal de poder-los comparar ràpidament, tenim els termes en una taula de hash. Utilitzem la següent estructura:

4.3. Fitxer Invers

Un fitxer invers és un mapeig de termes a documents. De cada terme, emmagatzem en quins documents ha aparegut i amb quina freqüència (de nou, sovint ens interessa guardar també en quines posicions apareix, per tal de poder cerca seqüències de paraules consecutives). Cada tupla {document, freqüència, posicions} s'anomena *posting* — en conjunt, formen una *posting list*.

4.4. Cerques per Semblança

4.4.1. Model *Term Frequency – Inverse Document Frequency (TF–IDF)*

El model *tf-idf* assigna pesos a cada terme $i \in T$ dins d'un document $d \in D$ en una col·lecció dins d'un document $d \in D$ en una col·lecció D (T és el *vocabulari* de D). El pes $w(i, d, D)$ assignat a cada terme és una aproximació de la *rellevància* del terme dins del document d, respecte la col·lecció D.

Així, anàlogament al model vectorial bàsic, cada document d és representat per un vector $\{w_1,..., w_{|T|}\}$, on cada w_i és el pes $w(i, d, D)$ assignat per *tf-idf*. Els pesos segueixen les següents definicions:

- Pes d'un terme i en un document d dins una col·lecció D de $|D|$ documents:

- $w(i, d, D) = tf(i, d) \cdot idf(i, D)$

- Freqüència relativa (*Term Frequency*) d'un terme i en un document d:
  - $tf(i, D) = \frac{f(i,d)}{\max_j f(j,d)}$

- Freqüència Inversa (*Inverse Document Frequency*) d'un terme i dins una col·lecció D de $|D|$ documents:
  - $idf(i, D) = \lg \frac{|D|}{df(i,D)}$

- Freqüència d'un terme i en un document d: $f(i, d)$

- Freqüència de documents (*document frequency)* d'un terme i en una col·lecció D — nombre de documents que contenen i: $df(i, D)$

Intuitivament, el model *tf-idf* assigna un pes major a termes molt freqüents dins un document, i un pes menor a termes que apareixen en molts documents. El factor *tf* serà 0 per termes amb freqüència zero i 1 pels termes amb la màxima freqüència. El factor *idf* serà $\lg |D|$ per termes que apareguin en un sol document i $\lg 1 = 0$ per termes que apareguin en tots.

4.4.2. Semblança Cosinus

*C*onsiderem que dos documents són *similars* si contenen pesos similars — donada la representació vectorial (de T dimensions) dels documents, els podem comparar geomètricament.

El mètode més habitual és la *semblança cosinus*, que consisteix en calcular el cosinus de l'angle que els separa. Això és equivalent al *producte escalar dels vectors normalitzats*.

$$Similarity(u, v) = \frac{u}{|u|} \cdot \frac{v}{|v|} = \hat{u} \cdot \hat{v} = |\hat{u}||\hat{v}| \cos \alpha = \cos \alpha$$

4.4.3. Algoritme

L'algoritme més fàcil per trobar els documents més similars a un altre consisteix en comparar-los un a un. Tanmateix, això tindria un cost prohibitiu per a col·leccions potencialment massives. En canvi, aprofitant que *només contribueixen al producte escalar aquells termes que apareixen en ambdós documents*, és habitual de fer el recorregut en sentit invers: si volem cercar documents similars a d, cerquem, per cada terme $t \in d$, els documents d_i en els quals apareix t utilitzant el *fitxer invers*.

A cada pas, anem acumulant els productes de pesos (components del vector) del document cercat d amb els altres documents d_i en una taula de semblances de documents. Partint de $\forall\, i : Similarity[d][d_i] = 0$, anem sumant $Similarity[d][d_i] \mathrel{+}= w(t, d) \cdot w(t, d_i)$.

A més, ens cal que els pesos estiguin normalitzats, és a dir, que $\sum_{\forall t} w(t,d) = 1$ per tots els documents. L'algoritme, en pseudocodi, és el següent:

```
def SimilarDocuments(d, inverted_file):
  similarities = {}
    for (t, w) in tf_idf(d):
      for d2 in inverted_file.posting_list(t):
        w2 = tf_idf(d2).t
        similarities[d2] += w*w2
  return similarities
```

4.5. Cerca Exacta de Frases

4.5.1. Expressions Booleanes

Donada una *query* en forma d'*expressió booleana* amb *conjuncions* (*AND,* '&'), *disjuncions* (*OR,* '|') i *negacions* (*NOT,* '!') sobre *paraules, conjunts de paraules ({...})* o *seqüències ("...")*, volem trobar aquelles frases que la compleixen, així com a quins documents apareixen. Podem definir el conjunt d'expressions vàlides amb la següent gramàtica incontextual

```
EXPR -> EXPR '|' TERM | TERM
TERM -> TERM '&' FACTOR | FACTOR
FACTOR -> SET | SEQ | '!' FACTOR | '(' EXPR ')' | WORD
SET -> '{' WORD* '}'
SEQ -> '"' WORD* '"'
```

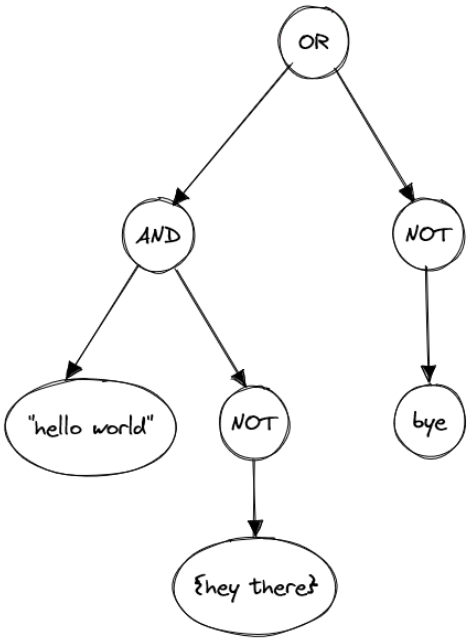
on WORD és qualsevol cadena de caràcters alfanumèrics no buida.

Tal com es dedueix de la gramàtica, la negació és la operació amb major prioritat, seguida de la conjunció i, finalment, la disjunció.

4.5.2. Representació – Parsing Tree

Per poder respondre la query, primer l'hem de convertir en un *parsing tree*: un arbre que descriu les operacions i l'ordre en què s'han d'executar. Els nodes fulla són paraules, conjunts o seqüències i les arrels d'algun subarbre són operacions. Per exemple, l'arbre corresponent a l'expressió

```
"hello world" & !{hey there} | !bye
```



és el següent:

Imatge 6. Parsing Tree de l'Expressió 'hello world & !{hey there} | !by'

4.5.3. Resolució de les Cerques

El *model vectorial simple* (i.e., amb freqüències enlloc de pesos *tf-idf*) amb *posicions* i *fitxer invers* ens serveix per resoldre les cerques. Les operacions *AND*, *OR* i *NOT* són equivalents, respectivament, a *interseccions*, *unions* i *diferències* de conjunts. A més, com que tenim les *posicions* en que apareix cada terme en cada documents, cercar una seqüència de paraules és com una intersecció però comprovant que les posicions siguin consecutives.

Per resoldre una expressió en forma d'arrel d'un arbre, hem de resoldre primer les subexpressions filles (recursivament) i ajuntar-les amb les operacions de conjunts convenients.

4.6. Implementació

4.6.1. BooleanExpression

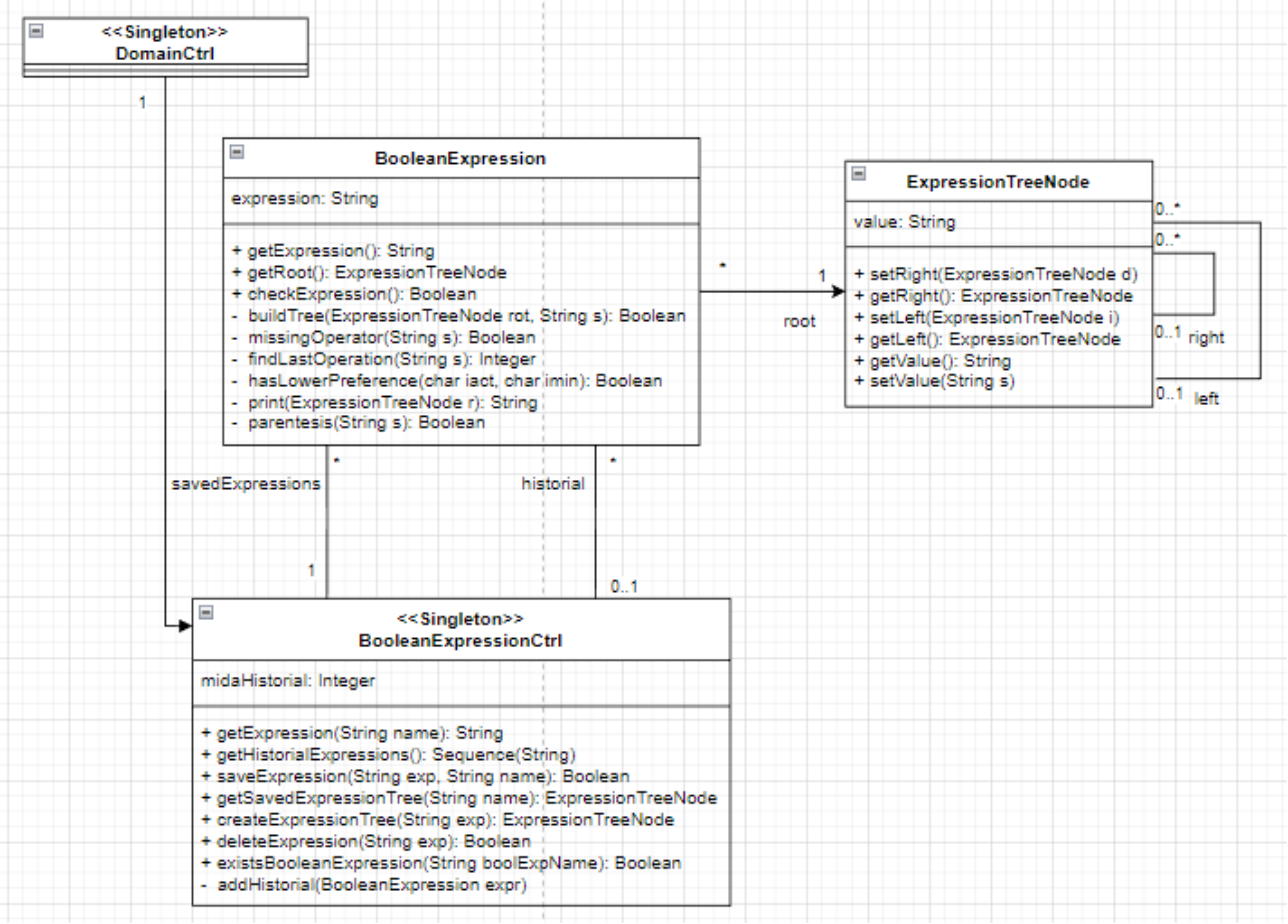
Representa una expressió booleana i conté una String on es guarda la expressió original i el node arrel de la representació en arbre d'aquesta. Només permet crear instàncies amb expressions correctes.

4.6.2. BooleanExpressionController

Conté totes les expressions booleanes guardades amb un nom i les gestiona, permet guardar expressions correctes i accedir a elles posteriorment. També s'encarrega de crear l'arbre de qualsevol expressió no guardada.

4.6.3. ExpressionTreeNode

És node d'un *bintree* normal i corrent que guarda un valor i un punter al fill dret i un al esquerre.



Imatge 7. Diagrama de Classes relacionades amb les Expressions Booleanes

4.6.4. Index

Hem vist que necessitem un *fitxer invers* i un *directe* tant de documents (per les cerques per semblança) com de frases (per les cerques booleanes). Per això, hem creat la classe *Index*, que emmagatzema ambdós fitxers (o índexs).

Per l'*índex invers* utilitzem la següent estructura

```
HashMap<String, HashMap<DocId, HashSet<Integer>>>
```

i.e., un mapeig de termes (utilitzem el tipus bàsic de java *String*) als documents (representats per un identificador – un tipus genèric) apareixen i en quines posicions. L'*índex directe* és similar, però a la inversa:

```
HashMap<DocId, HashMap<String, HashSet<Integer>>>
```

La classe té un mètode per inserir documents, *Index::insert(docId, content),* i per eliminar-los, *Index::remove(docId)*. A més, té força mètodes de consulta. Descriurem els essencials – n'hem afegit més que treballen sobre aquests, per facilitar la feina a les classes que l'utilitzen.

- *Index::postingList(term)*: retorna la *posting list* (un `HashMap<DocId, HashSet<Integer>>`) corresponent al terme indicat, o un mapa buit si el terme no hi és.
- *Index::positions(docId)*: retorna el mapa de termes i posicions corresponents o un opcional buit en cas que el document no hi sigui. I.e., retorna `Option.some(HashMap<String, HashSet<Integer>>)`si el document és present o `Option.none()` sinó.

  És important destacar que, en aquest cas, no podem retornar un mapa buit – no podríem distingir-lo d'un document buit.

4.6.5. BooleanModel

Per a emmagatzemar els índexs de frases i resoldre les cerques booleanes hem escrit la class *BooleanModel*. És bàsicament un *wrapper*, ja que l'únic membre que conté és un *Index* de les frases. A més, a part de mètodes de modificació (que deleguen la feina a l'*Index*), té un seguit de mètodes de consulta per a resoldre les operacions finals de les expressions booleanes:

- `BooleanModel::queryTerm(term):` retorna el conjunt de documents on apareix el terme.
- `BooleanModel::querySet(terms):` retorna el conjunt de documents on apareixen tots els termes. És equivalent a la intersecció dels resultats de `queryTerm` per cada terme.
- `BooleanModel::querySequence(terms):` retorna el conjunt de documents on apareixen tots els termes, en l'ordre indicat.
- `BooleanModel::all():` retorna tots els documents de l'*índex*. És necessari per fer negacions, utilitzant que $\overline{S} = \Omega - S$, on Ω és l'univers, i.e., tots els documents.

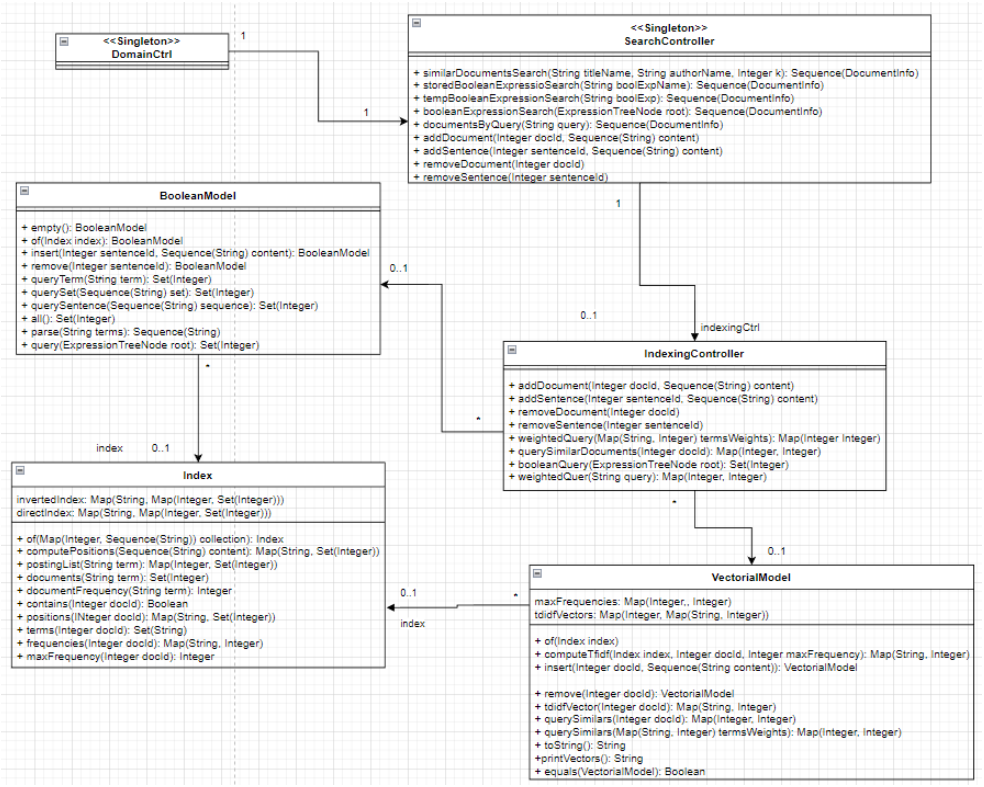
4.6.6. VectorialModel

Hem creat la classe *VectorialModel* per emmagatzemar els índexs de documents i realitzar cerques de documents per similaritat. A part de la instància d'*Index* corresponent, també guarda la freqüència màxima de cada document (ho necessitem per calcular ràpidament els pesos *tf-idf*), així com els seus vectors *tf-idf*.

Podem cercar per semblança de dues maneres diferents. D'una banda, a partir de la *id* d'un document contingut en l'índex. De l'altra, a partir d'una *query* representada per un vector de termes i pesos – fem servir un `HashMap<String, Double>`.

4.6.7. Indexing Controller

El conjunt de funcionalitats i dades dels models estan agrupats en l'*IndexingController*. Aquesta classe s'encarrega d'emmagatzemar ambdós així com de realitzar les crides necessàries per resoldre les consultes i dur a terme les modificacions requerides.



Imatge 8. Diagrama de Classes relacionades amb la Indexació

5. Preprocessament

El preprocessament és la porta d'entrada al tractament dels documents. Consisteix a convertir els fitxers crus (tires de caràcters) a llistes de termes. Particularment, ens interessa colapsar el text al mínim nombre de ***tipus*** (o termes) representatiu; volem minimitzar el nombre de termes diferents sense perdre informació.

Per exemple, convertim totes les paraules a minúscules (de forma que "Casa", "casa" i "CASA" es converteixen en un sol terme), eliminen declinacions ("casa", "cases" i "caseta") i **stopwords** (preposicions, articles, etc. molt comuns que no aporten significat semàntic).

5.1. Tokenització

Abans de tot, ens cal convertir els documents d'entrada en llistes de termes (i.e., ***tokens***), la qual cosa no és trivial. Per exemple, segurament volem eliminar els punts a final de frase; però, i els punts dins d'una URL? També s'ha de pensar què fer amb apòstrofs, guions, símbols no-ascii, etc.

El més senzill (i agressiu) és tallar les paraules a qualsevol caràcter no alfabètic, però també existeixen tokenitzadors més sofisticats que reconeixen URLs i adreces, etc.

5.2. Stopwords

És comú, en el món de l'*information retrieval*, de filtrar les anomenades *stopwords*: aquelles paraules que tenen freqüència molt alta i que no aporten significat semàntic (e.g., articles, pronoms, verbs auxiliars, etc.)

5.3. Stemming

La *lemmatització* és el procés consistent a convertir tots els termes als seus lexemes. Tanmateix, això és a la pràctica molt complicat, donada l'abundància de formes irregulars, arrels similars amb significats diferents, etc.

Per això, s'aproxima amb un algoritme de *stemming*, que colapsa força bé diferents termes a una mateixa arrel. Ara per ara, tanmateix, no ho hem implementat – queda, possiblement, com a feina futura.

5.4. Implementació

5.4.1. Tokenizer

Només conté dues funcions: *splitSentences* – que de moment no s'utilitza ja que prenem les frases una a una per consola – i *tokenize*, que ara per ara simplement talla les paraules als espais en blanc.

5.4.2. Stopwords

Llegeix les llistes d'*stopwords* proporcionades i ofereix el mètode *filter(term)*, que retorna cert si i només si el terme no és una *stopword*.

5.4.3. TokenFilter

És el punt d'accés al filtratge de tokens. Té un únic mètode estàtic, *filter(term)*, que retorna opcionalment el terme filtrat. Ara per ara (ho extendrem si acabem fent un *stemmer*, per exemple), converteix les paraules a minúscula i filtra les *stopwords*.

6. Altres classes

A continuació descrivim aquelles classes secundàries que no encaixaven en altres apartats. Es tracta, en general, de classes amb només mètodes estàtics.

6.1. Relacionades amb la Indexació

6.1.1. Parsing

Conté algunes funcions que hem fet servir per llegir documents del sistema de fitxers, a fi de provar el sistema amb corpus grans. No les expliquem ja que no formen part de la primera entrega.

Sí que és rellevant, però, el mètode *weightedQuery(string)*, que *parseja* una query amb pesos (indicats amb l'operador *boost*, '^', seguit del pes que es vol atribuir – per exemple, una possible

query seria 'hola^2 adeu^0.5') i retorna, si la query és correcta, un vector de termes i pesos (`HashMap<String, Double>`). Sinó, retorna un String descrivint els errors de format.

6.2. TFIDF

Conté un sol mètode estàtic *computeTFIDF(termFrequencies, maxFrequency, documentsCount, documentFrequencies*), que retorna un vector de termes i pesos *tf-idf* del document, dins de la col·lecció, descrits pels paràmetres.

6.3. Helpers

El package *helpers* conté algunes funcions genèriques. Està classificat pel context en què s'utilitzen – les classes són: *Functional, Maths, Sets* i *Strings*.

7. Apèndix

7.1. Autors de les Classes

7.2. Domain

7.2.1. Controllers

- BooleanExpressionCtrl: Xavier
- DocumentCtrl: Biel
- DomainCtrl: Xavier
- SearchCtrl: Xavier
- SentenceCtrl: Biel
- TitleCtrl: Xavier
- AuthorCtrl:Xavier

7.2.2. Core

- Document: Biel
- DocumentInfo: Biel
- Sentence: Biel
- Title: Biel
- Author: Biel

7.2.3. Expressions

- BooleanExpression: Jan
- ExpressionTreeNode: Jan

7.2.4. Indexing

- Boolean
  - BooleanModel: Marcel. En Jan ha fet la part de resoldre queries booleanes
- Core
  - Index: Marcel
  - Parsing: Marcel
  - IndexingController: Marcel

- Vectorial
    - TFIDF: Marcel
    - VectorialModel: Marcel

7.2.5. Preprocessing

- Stopwords: Marcel
- TokenFilter: Marcel
- Tokenizer: Marcel

7.3. Helpers

Marcel

7.4. Driver

Xavier

7.5. Llibreries Externes

- Vavr