

# Documentació 3r Entrega

## GRUP 42.2

UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

---

Facultat d'Informàtica de Barcelona



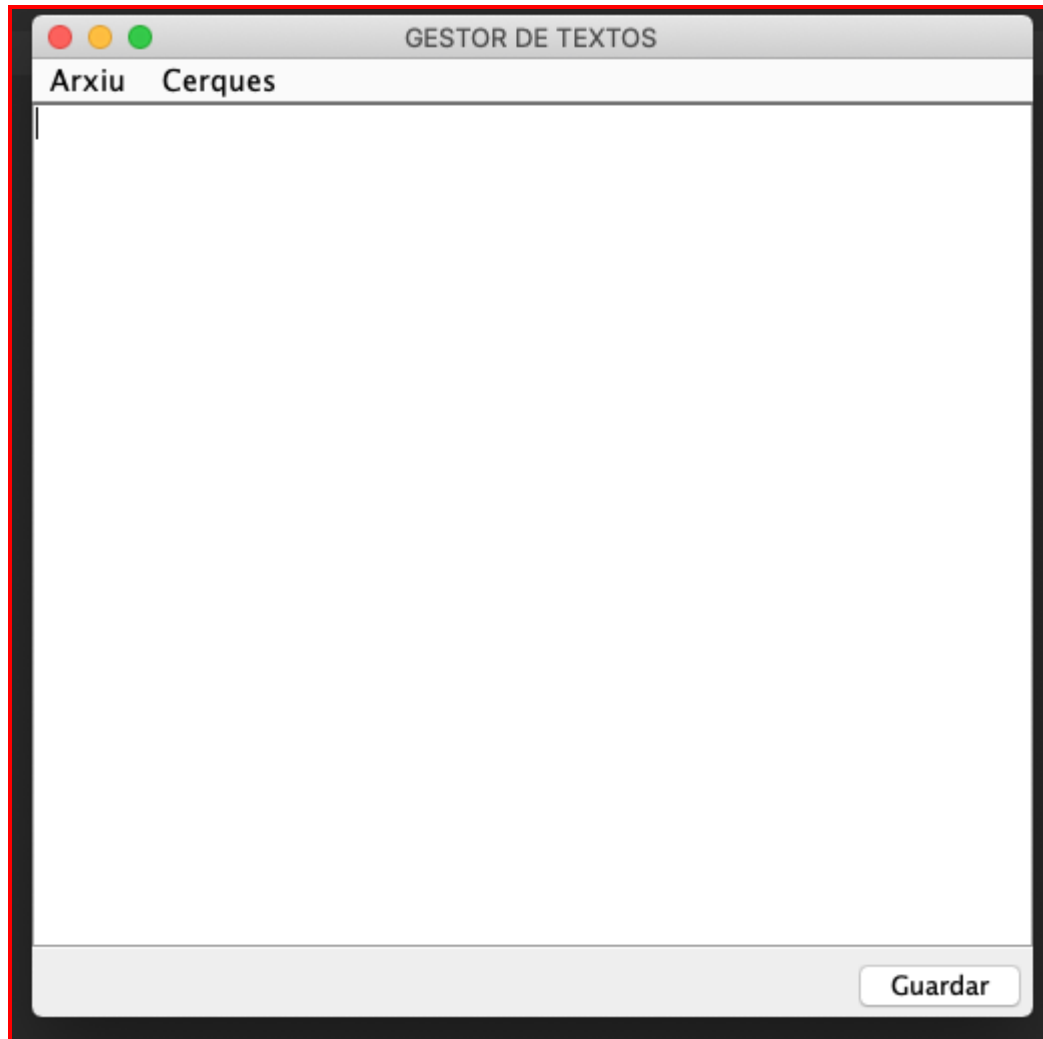
Nora Caballero de Llanos  
Lluc Hospital Escat  
Richard Pie Sánchez  
Ruben Rivera Villoldo

# ÍNDEX

<b>1. Manual D'Usuari</b>	<b>2</b>
<b>1.1. Menú Principal</b>	<b>2</b>
<b>1.2. Arxiu</b>	<b>3</b>
1.2.1. Carregar Document	4
1.2.2. Obrir Document	5
1.2.3. Eliminar Document	6
1.2.4. Recuperar Document	6
<b>1.3 Cerca</b>	<b>7</b>
1.3.1 Expressió Booleana	8
1.3.2 Contingut Document	10
1.3.3 K Documents Semblants	11
1.3.4 Títols de l'Autor	12
1.3.5 Autors per Prefix	13
<b>2. Estructures de dades i Algorismes</b>	<b>14</b>
2.1 Estructures de dades globals	14
2.2 Algorisme Espai Vectorial+Similitud per cosinus	18
2.2.1 Realització:	18
2.2.2 Estructura de dades utilitzades:	19
2.3 Algorisme cerca per Booleans	20
2.3.1 Realització:	20
2.3.2 Estructures de dades utilitzades:	21
<b>3. Repartiment de classes</b>	<b>22</b>
<b>4. UML</b>	<b>23</b>
4.1 Diagrama Presentació	23
4.2 Diagrama Domini	24
4.3 Diagrama Persistència	25

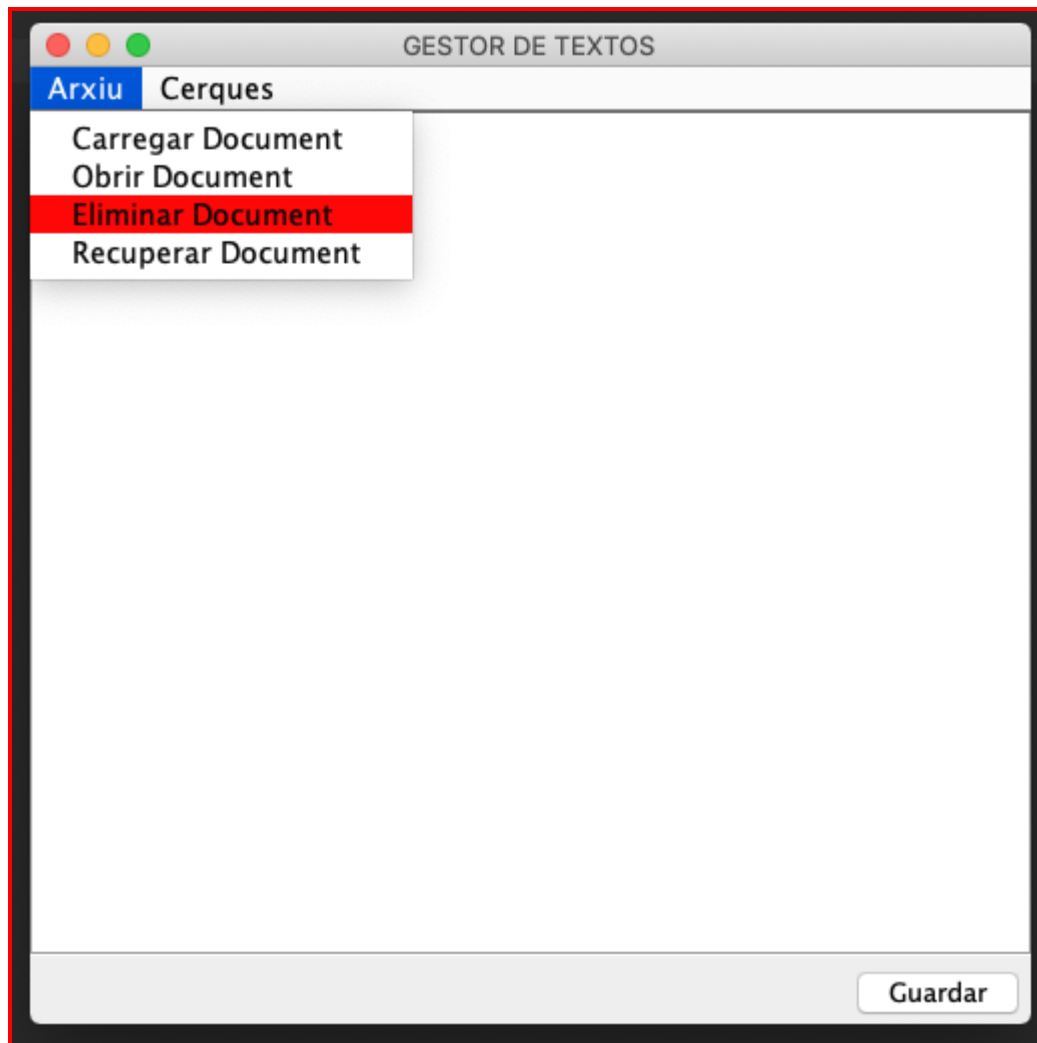
# 1. Manual D'Usuari

## 1.1. Menú Principal



Des del menú principal és des d'on es podran dur a terme totes les funcionalitats del programa, dividides principalment en dues seccions: “Arxiu”, que s’encarregarà del processament de documents del nostre sistema; i “Cerques”, que contindrà totes les operacions de cerca dels documents guardats al nostre programa.

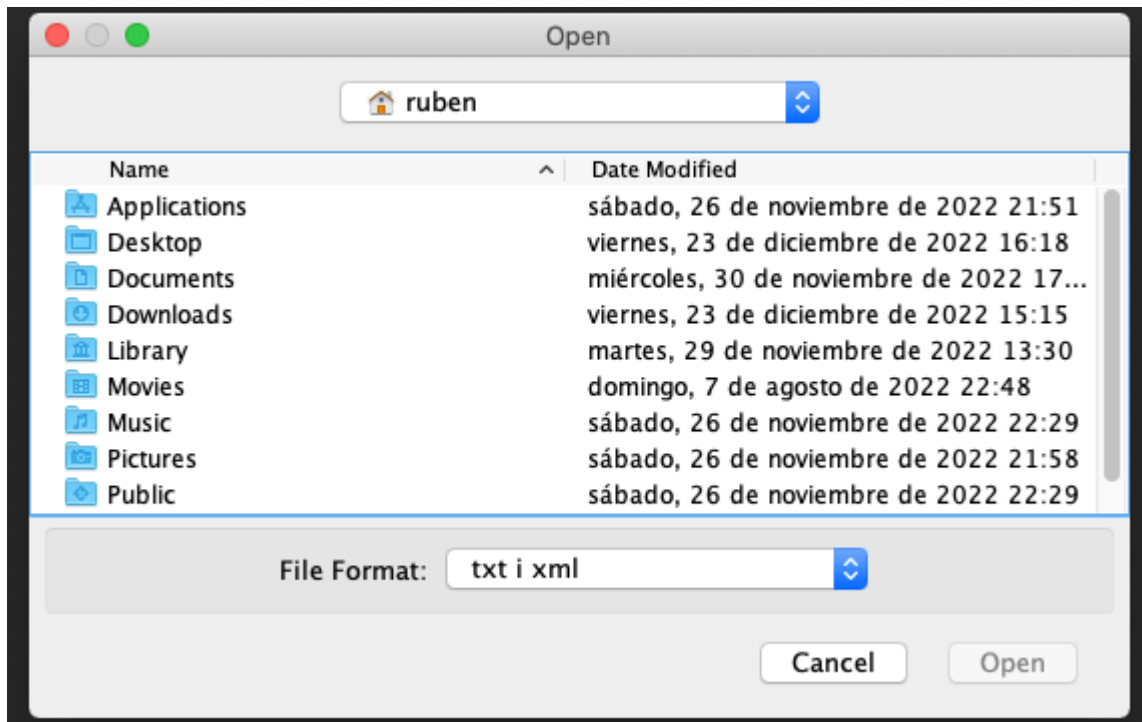
## 1.2. Arxiu



En el menú d'Arxiu es on es mostren les funcionalitats principals de documents:

- Carregar Document: aquesta funcionalitat permet carregar un document a la carpeta
- Obrir Document: permet obrir un document que es troba guardat a la carpeta
- Eliminar Document: permet eliminar un document de la carpeta i aquest es guarda a la carpeta eliminats
- Recuperar Document: permet recuperar un document eliminat amb anterioritat i es guarda a la carpeta.

### 1.2.1.Carregar Document



Un cop cliquem l'opció de Carregar Document, s'obrirà una finestra, que ens ubicarà al directori de l'usuari en el qual estem, que ens permetrà seleccionar el document a importar al nostre sistema. Un cop seleccionat el document que volem, en pitjar el botó Open es carregarà a la nostra carpeta estàndard de documents.

Quan vulguem fer gestions dels documents carregats a la nostra carpeta, s'obrirà un selector de document que ens donarà dues llistes desplegable. La primera ens fa seleccionar quin dels autors que hi ha al nostre sistema volem escollir, mentre que la segona ens permet triar un dels títols de l'autor seleccionat anteriorment. Les opcions que treballen amb aquest selector de document són Obrir Document, Eliminar Document, i Recuperar Document.



### 1.2.2. Obrir Document

A l'opció d'obrir document, es mostrarà el contingut del document escollit a la finestra de Selecció de document, sigui per lectura, o per modificar el seu contingut.

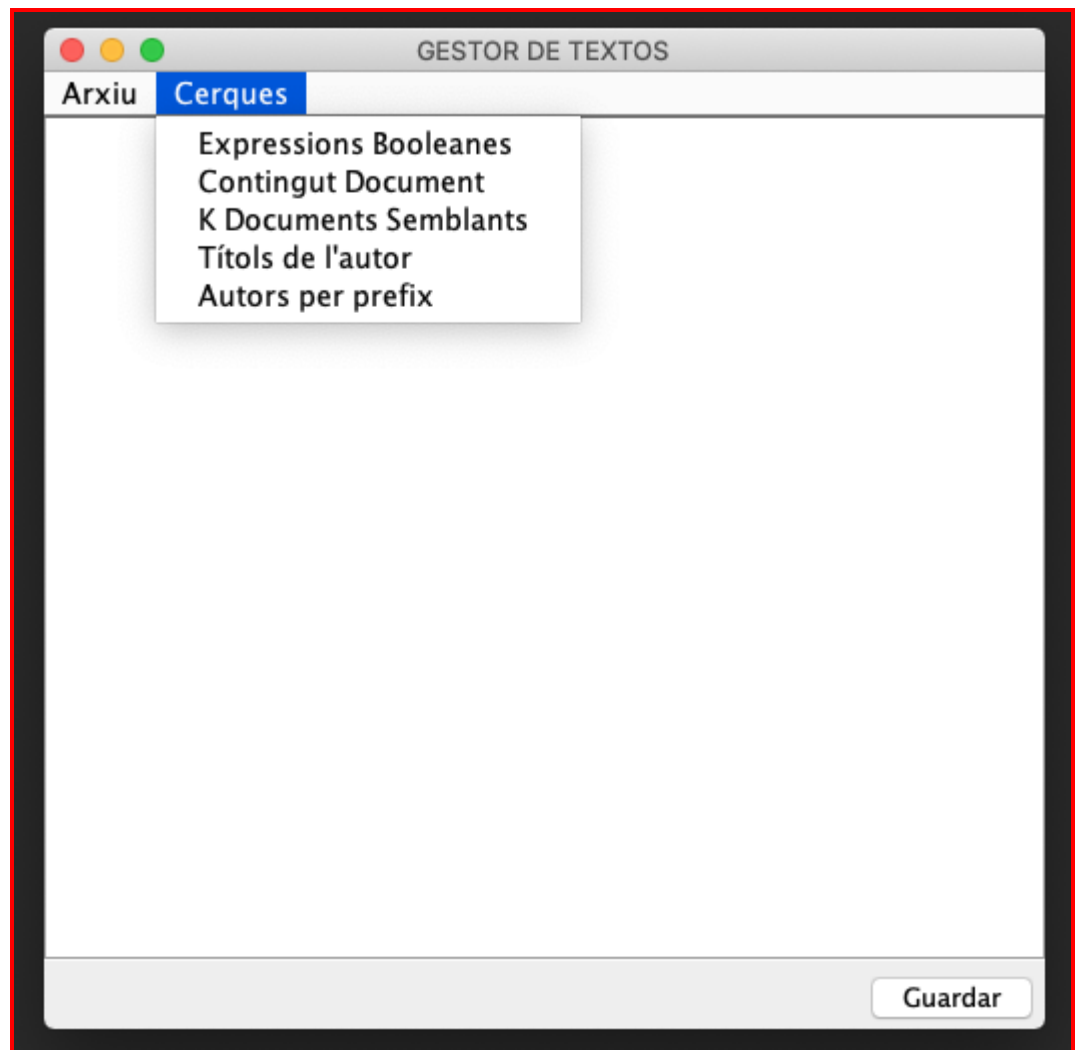
### 1.2.3. Eliminar Document

El document seleccionat es mourà a una carpeta d'eliminats, on romandrà fins que es tanqui el programa. En el cas que hagi comès un error en l'eliminació del document, l'usuari també té l'opció de recuperar el document en temps d'execució.

### 1.2.4. Recuperar Document

En el cas de voler tornar a carregar un document que s'ha eliminat durant l'execució del programa, l'usuari pot seleccionar el document des de la carpeta de documents eliminats i es traslladarà de nou a la carpeta estàndard, des d'on podrà fer la resta de funcionalitats del programa amb ell. És important remarcar que, en el cas que un document es trobi a la carpeta de documents eliminats, i finalitzi l'execució del programa, el document es perdrà definitivament al nostre sistema.

## 1.3 Cerca

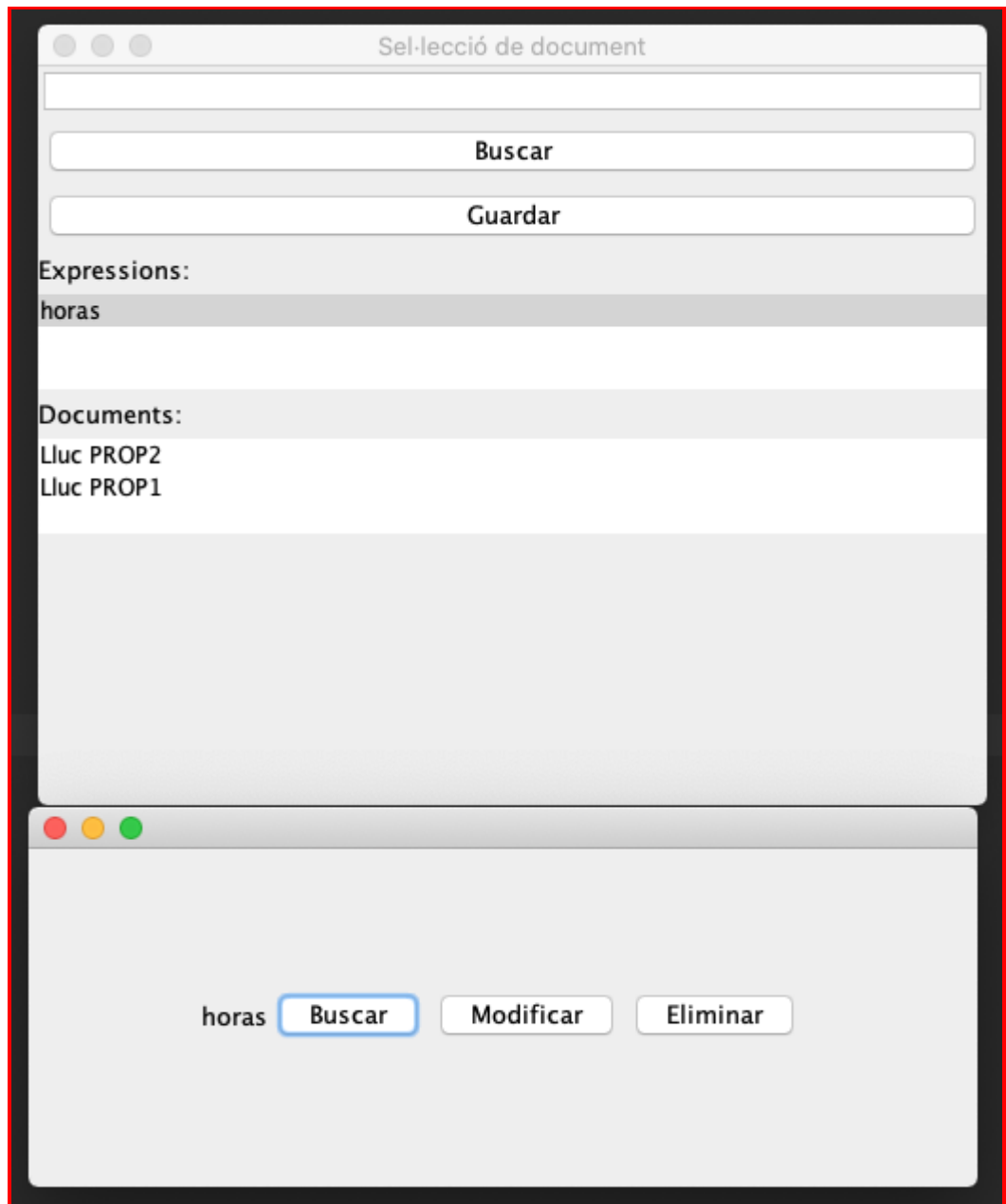


En el menú de Cerca és on estan situades les funcionalitats principals de cerca:

- **Expressió Booleana:** permet gestionar les expressions Booleanes i buscar-les als documents
- **Contingut Document:** permet llegir el contingut d'un document
- **K Document Semblants:** permet llistar x documents semblants a un document
- **Títols de l'Autor:** permet llistar els títols dels fitxers d'un autor
- **Autors per prefix:** permet llistar els autors donat un prefix.



### 1.3.1 Expressió Booleana

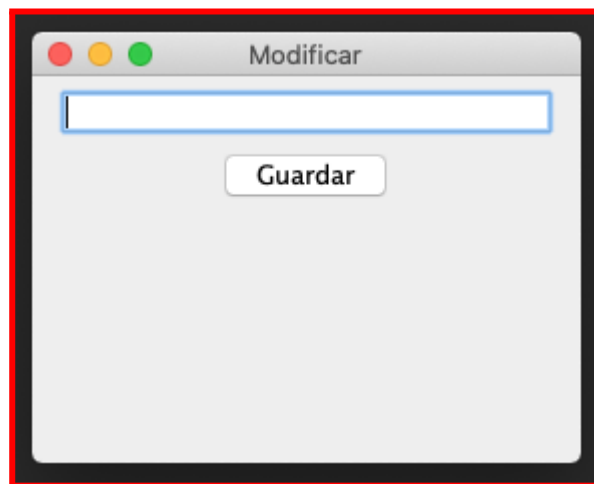


La pantalla principal d'aquesta funcionalitat, permet guardar i buscar expressions booleans. Com es pot veure la pantalla consta d'un espai per introduir text, dos botons (buscar i guardar) i dues llistes: una d'expressions i l'altre de documents.

Per afegir una expressió booleana el sistema, primerament s'ha d'introduir l'expressió a l'espai corresponent i premeu el botó de guardar. En el cas que l'expressió no estigui guardada prèviament, aquesta es guardarà, en cas contrari apareixerà un error informant que l'expressió ja forma part del

sistema. Si no hi ha hagut el missatge d'error, a continuació es mostra la nova expressió afegida a la llista d'expressions.

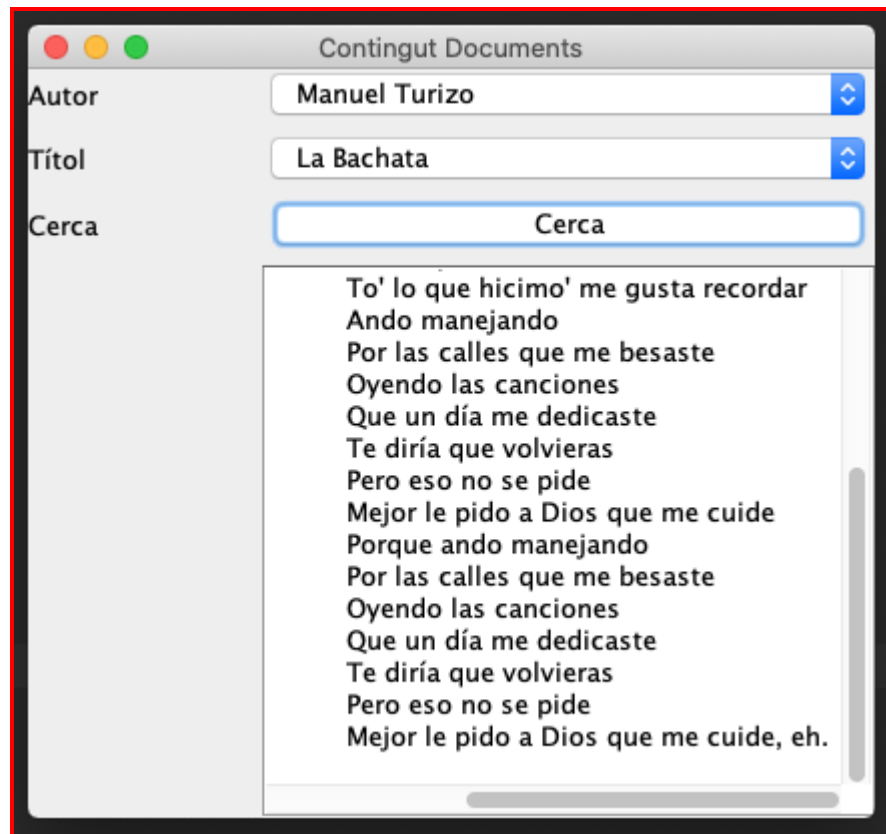
En el cas de voler modificar, eliminar, buscar una expressió de la llista, s'ha de seleccionar l'expressió. Seguidament, apareixerà una pantalla secundària on es pot indicar l'opció que es vol utilitzar. En el cas de voler modificar l'expressió apareixerà una altra pantalla on permet introduir la nova expressió i pitjar el botó Guardar. Un cop premut el botó de canviar la pantalla de modificar es tanca automàticament.



Si es vol eliminar l'expressió, en pitjar el botó Eliminar, la pantalla actual es tanca. I l'expressió eliminada desapareix de la llista d'expressions de la pantalla principal de cerca expressions booleanes.

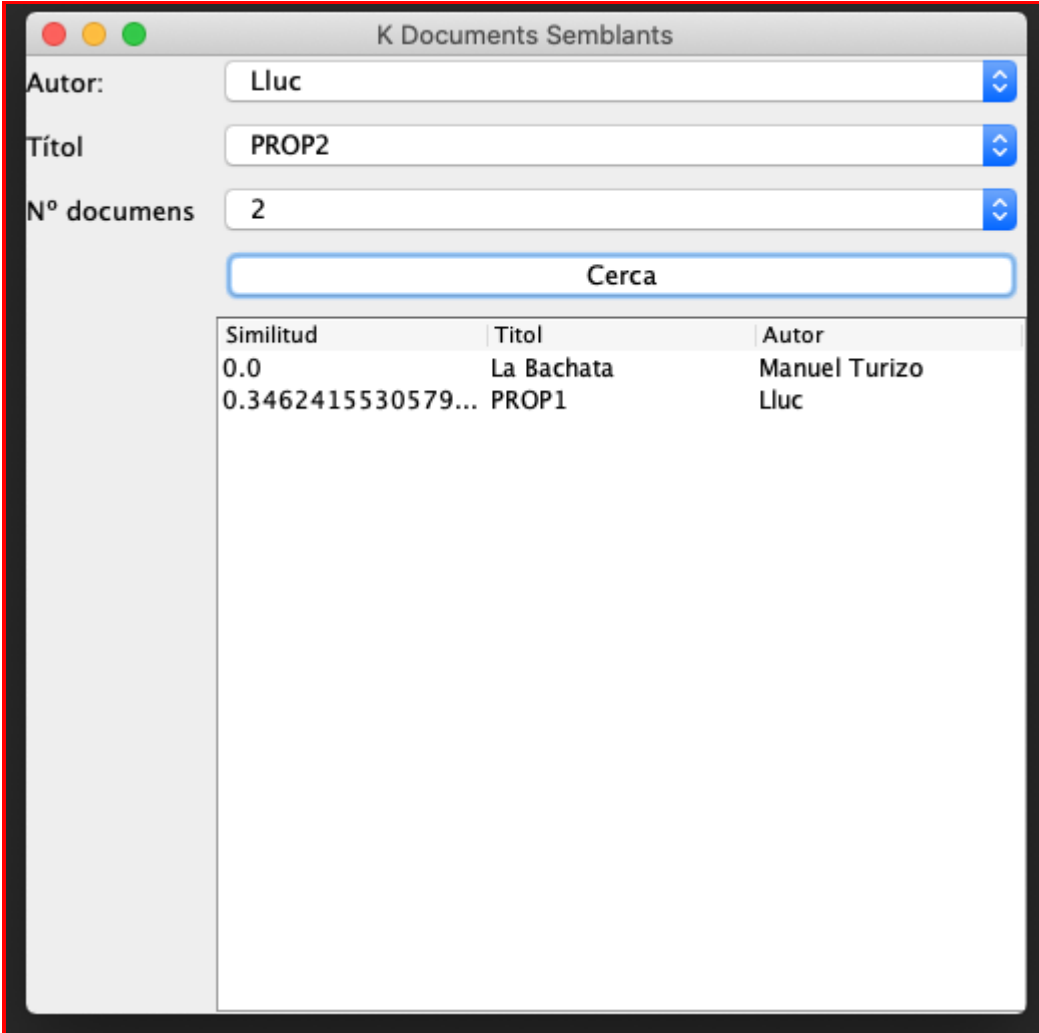
Per poder buscar, s'ha d'introduir l'expressió al lloc corresponent i pitjar el botó de Buscar, seguidament apareixerà a la llista de Documents: els documents que compleixin amb les condicions de l'expressió booleana.

### 1.3.2 Contingut Document



Per poder veure el contingut d'un document, primerament s'ha de seleccionar un autor dins del llistat que es desplega en clicar sobre rectangle situat al costat de "Autor" i fer el mateix procediment per seleccionar el títol, clicar sobre el rectangle situat al costat de "Títol" i seleccionar un dels títols. Finalment, en clicar sobre el botó de Cerca, es mostrarà el contingut del document seleccionat. El document només es podrà llegir, en cas de voler modificar-lo has d'accedir pel menú arxiu i obrir-lo. El menú arxiu està situat a la pantalla principal del sistema.

### 1.3.3 K Documents Semblants

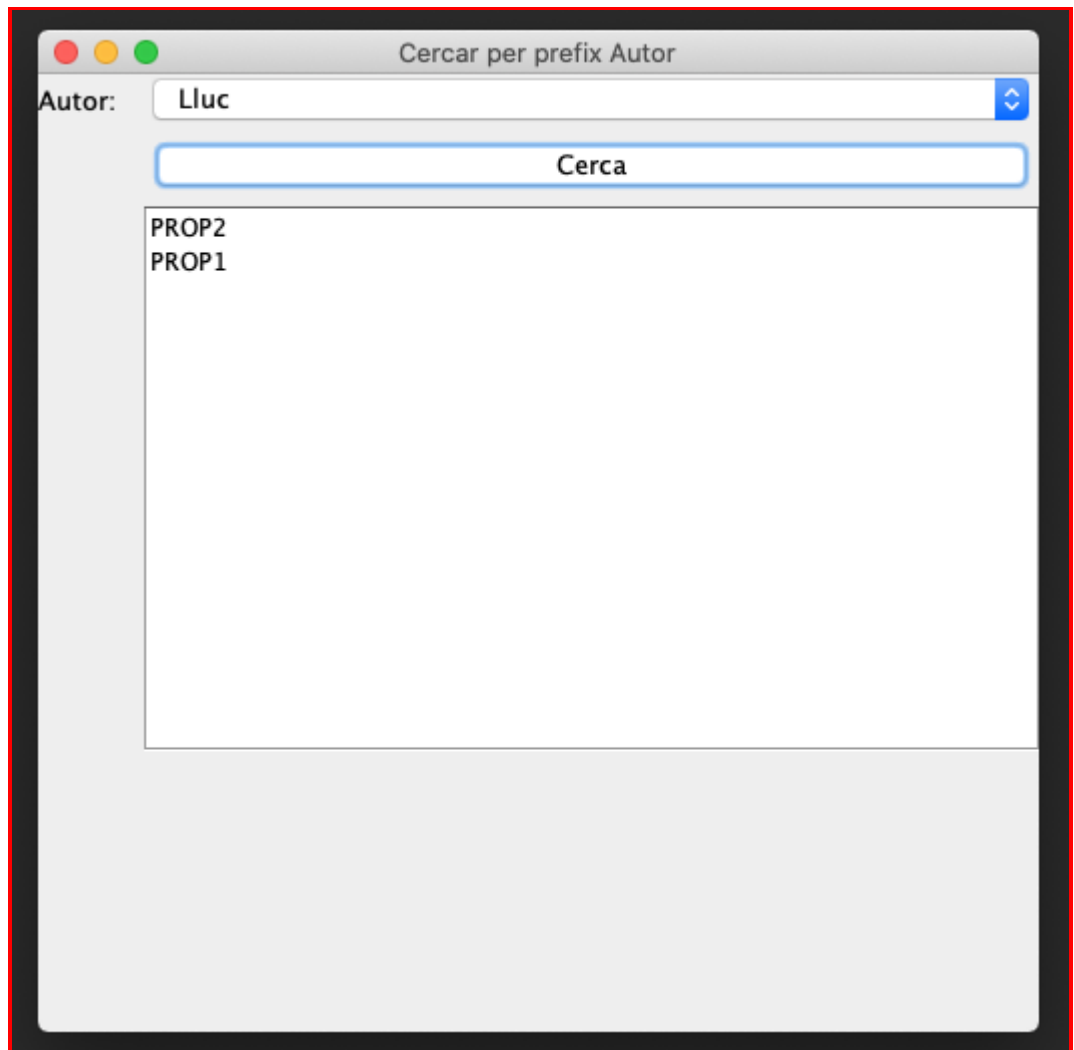


Similitud	Títol	Autor
0.0	La Bachata	Manuel Turizo
0.3462415530579...	PROP1	Lluc

La funcionalitat que permet fer aquesta pantalla és la de llistar documents. Llista els documents seguin un mètode que donat un document i un número x, llista x documents més semblants al document donat.

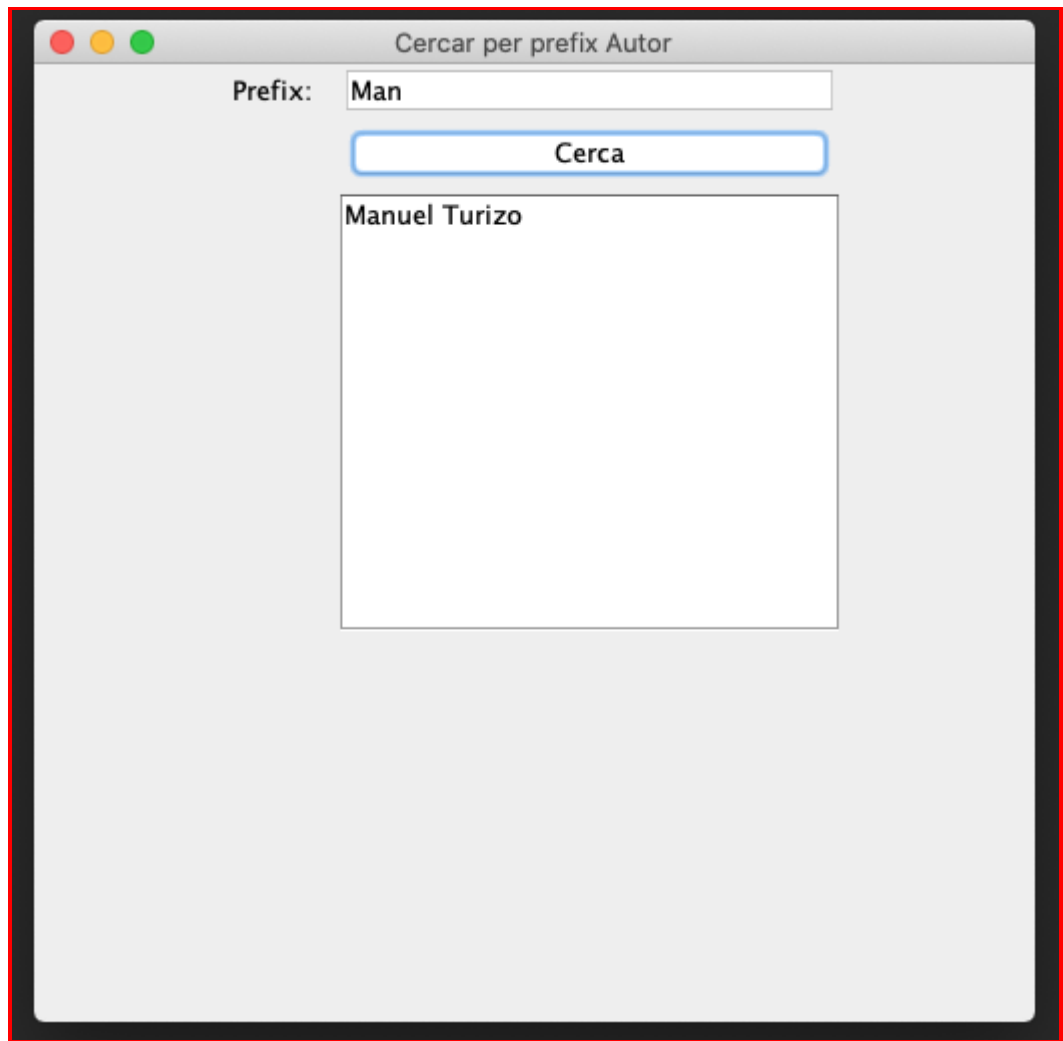
Per fer això, has d'indicar primerament el document. Per indicar el document necessites indicar l'autor i el títol. Per indicar l'autor has de pitjar al requadre situat al costat de "Autor:" i seguidament es desplegarà un llistat dels autors disponibles on es pot seleccionar el desitjat. A continuació has de repetir el procés amb títol, pitjant el rectangle situat al costat de "Títol" i seleccionar de la llista el títol desitjat. Un cop seleccionat l'autor i el títol s'ha de seleccionar el número de documents que vols rebre. A continuació s'ha de pitjar el botó de Cerca i seguidament es llistaran els documents.

### 1.3.4 Títols de l'Autor



Aquesta pantalla ens permet fer la cerca dels títols d'un autor. Per poder fer la cerca primer s'ha d'indicar l'autor dels títols que es volen buscar. Per indicar-ho primerament s'ha de clicar el rectangle situat al costat de "Autor". A continuació es desplegarà un llistat dels autors dels quals hi ha algun document seu al sistema. En seleccionar un autor de la llista i a continuació pitjar el botó de cerca, es llisten a sota els títols de tots els documents de la carpeta que siguin de l'autor seleccionat.

### 1.3.5 Autors per Prefix



Aquesta pantalla ens permet utilitzar la funcionalitat de cercar autors donat un prefix.

Per fer això cal indicar el prefix en el rectangle situat al costat de "Prefix:", un cop introduït i en pitjar el botó de Cerca. A continuació es llisten tots els autors els quals tenen algun document a la carpeta, on el seu nom comença pel prefix introduït.

## 2. Estructures de dades i Algorismes

Primer de tot el que farem un repàs de les principals estructures de dades utilitzades en el domini per dur a terme les funcions de negoci demanades de la manera més eficient possible. Tot seguit passarem a explicar els algorismes emprats amb les seves estructures de dades pertinents i el perquè de la utilització d'aquestes i no d'altres opcions que també veurem.

### 2.1 Estructures de dades globals

**TreeMap<String,TreeMap<String, Document>> Carpeta:**

Key: autor

Value:(Key: Títol, Value: Document com a objecte)

En aquest cas el que volíem era una estructura de dades per guardar el conjunt d'objectes documents de manera ordenada segons autor i títol, a més a més de poder tenir una manera d'identificar-los única. Vam optar per un TreeMap, ja que ens va semblar la manera més eficient, ja que es tractava d'una estructura on hauríem de realitzar moltes consultes, eliminar, inserir... i aquesta ens ho permetia en costos de temps baixos. Si mirem concretament els costos i característiques del TreeMap trobem el següent:

- Tenim ordenats els autors i títols en ordre natural de String, és a dir, en ordre alfabètic, de manera que a l'hora de treure títols, autors..., ho tenim ordenat com es veu millor per l'usuari.
- Cerca, inserció, get:  $\log(n)$ . Per tant en el nostre cas en temps mitja tarda ( $\log(n)+\log(m)$ ) en el cas que volguéssim l'objecte document o  $\log(n)$  en cas que volguéssim títols.

- La possibilitat de tenir el document identificat per autor i títol gràcies als `get()` i no de tenir un identificador i després comprovar que sigui aquell document.
- La possibilitat de tenir tanta memòria com necessitem.

Però si ho comparéssim amb altres possibilitats com per exemple:

- `HashMap`, veiem que pel que fa a costos és més eficient, ja que el `HashMap` per aquestes funcionalitats esmentades abans amb condicions ideals tenen un cost de (1). Però el problema el trobem a l'hora de mantenir un ordre, ja que aquesta estructura no està ordenada. Per això ens quedem amb el `TeeMap`, ja que si no en certes geters del domini (títols, autors, ...) que volem que surtin ordenats ho haurem de fer manualment cosa que afegiria un cost temporal i ens ho faria més ineficient.
- `Vector`, sabem que el seu cost mitjà en les funcionalitats anteriors, si les realitzem de manera seqüencial, és de (n), per  $n = \text{longitud del vector}$ . Per tant, no ho veiem eficient en comparació a un `TreeMap`. Tot i això pel que fa a costos podria ser més eficient si realitzéssim, per exemple cerques binàries, però la qüestió que hi hem trobat és el fet que el vector ha d'estar ordenat, cosa que no ocorre en el nostre cas, per tant, també vam descartar aquesta opció.

### **List<Type> x:**

Tot i ser una estructura bàsica l'hem utilitzat en molts casos per tal de more volums de dades únics, sobretot en retornar títols, autors, paraules... En aquest cas hem volgut emprar llistes en comptes de vectors no tant per la seva eficiència, ja que en molts casos el vector seria més eficient utilitzant segons quins algorismes d'inserció i cerca, sinó per la incertesa del tamany d'aquest, perquè a un vector hi ha la necessitat d'inserir una mida encara que es pugui incrementar. A més a més, també hi ha més comoditat a l'hora de tractar dades amb llistes que amb vectors.

### **Vector<String> expr:**



En aquest cas el que volem aquí és guardar totes les expressions. El perquè d'utilitzar un vector el trobem en l'eficiència donada per aquesta estructura envers les estructures més properes com podrien ser un set o una llista. El que volem aconseguir és una estructura de dades que ofereix un temps constant per molts elements que hi hagi (ja que la gent sol guardar i no esborrar, podent provocar ineficiència per molts elements) i que a més a més sigui adequat per a operacions de lectura i escriptura ràpides. Per aquestes raons hem escollit el vector en comptes d'un set o una llista, sobretot per la variabilitat de cost temporal segons el nombre d'elements.

**HashMap<String, Integer> tf\_hash;**

Key: Type

Value: Freqüència de la paraula en el document

**HashMap<String,Document> posting\_list;**

Key: Type

Value: Documents en el qual apareix la clau

Aquestes estructures les hem utilitzat per a dur a terme els nostres algorismes que es contenen seguidament d'aquesta breu explicació. En aquest cas el que hem buscat és una estructura de dades ràpida respecte a cost temporal, ja que es tracta d'un algorisme i volem que sigui el més eficient possible. Per això hem decidit les taules de Hash que tenen les característiques següents:

- Elements no ordenats, com no volem que els elements estiguin ordenats en tots moments i si ho han d'estar no és per la clau, ens estalviem el fet d'ordenar cada cop que es borren o s'insereixen elements i per tant fem la feina més eficient.
- Ocupa menys espai que altres estructures com el TreeMap, i a l'hora de guardar-lo en és més còmode.
- Costos d'operacions bàsiques (get(), put(), contains(), remove(), containsValue(), size(), isEmpty(), clear()): Tenen un cost mitjà constant  $\theta(1)$ , això si, tenint en compte certs factors:
  - La capacitat inicial no pot ser molt baixa, si no és més costosa segons quines operacions (en el nostre cas es posen forces paraules en ser textos, per tant, només passaria al principi)

- La càrrega no és molt elevada, cosa que no passa a no ser que posem una quantitat de textos molt gran o els textos tinguin molts milers de paraules no repetides.
- Segons la funció usada de Hash, pot provocar lentitud en segons quines condicions. En el nostre cas la funció utilitzada és hashCode(), una funció molt eficient en Java sempre i quan les classes de les claus siguin classes amb autogestió de la funció (en el nostre cas String, per tant si), que la distribució de dades sigui mínimament estable (en el nostre cas no podem saber, dependrà de la freqüència de les paraules, però sol ser força estable tenint en compte que es treuen les més freqüents) i el mida de la taula de hash.

Aquesta és l'opció que hem escollit, però si mirem altres possibles opcions veurem perquè és la millor per la feina que és dur a terme.

- TreeMap, és una opció vàlida, però té diferents qüestions que ens tiren enrere. La primera és el fet del cost de les operacions bàsiques, és de  $\log(n)$ , tot i que pot variar com hem explicat prèviament en l'apartat de TreeMap. El segon fet és que el TreeMap ordena els valors per clau, cosa que nosaltres no volem, per tant, quan hàgim d'ordenar per valor estariem fent la feina dos cops, cosa ineficient, incús en el cas de la `posting_list` no tenim necessitat d'ordenar els elements, per tant, no seria una opció vàlida.
- Vector, Set, List, ens trobem amb un problema i és que n'obliguen a emmagatzemar els elements específicament. A més a més, al no ser estructures de clau valor ens obliguen a crear una classe pair per guardar la clau-valor i segons quines operacions acabarien essent ineficients (seqüencial:  $(n)$ , no seqüencial el cost dependria de la funció utilitzada). Hauríem usat aquesta opció si haguéssim necessitat accessos aleatoris als elements, però en aquest cas hi havia necessitat de fer moltes cerques específiques, fet que hauria provocat força ineficiència.

## 2.2 Algorisme Espai Vectorial+Similitud per cosinus

Aquests dos algorismes ens serveixen per poder calcular la similitud entre un document seleccionat i la resta de documents del conjunt, de tal manera podrem saber quins documents són més similars o menys similars al document que indiquem. Per realitzar això calcula el tf-idf dels documents, per mesurar el pes de les paraules sobre un document tenint en compte el conjunt. Un cop realitzat aquest càlcul realitzant la similitud per cosinus podem saber la semblança dels documents. Seguidament veurem com s'ha implementat amb pseudocodi l'algorisme en el nostre programa:

### 2.2.1 Realització:

#### 1.Omplir tf\_hash i i posting\_list per cada document:

```
for(paraula in document){
    if(paraula not exist in tf_hash){
        afegir a tf_hash(paraula, 1)
        if(paraula not exist in posting_list){
            afegir a posting_list(paraula, document)
        }
        else{
            afegir document a posting_list.get(paraula)
        }
    }
    else{
        incrementar 1 a tf_hash.get(paraula)
    }
}
```

#### 2.Càlcul de vector de pesos de paraules per cada document (tf-idf)

```
for(paraula in posting_list){ //per cada type del conjunt de documents
    tf = freqüència paraula en el document(si !exist f = 0)/màx f del document
    idf = log2(nombre de documents/n de documents que contenen la paraula)
    vector.put(paraula, tf*idf)
```

```
}
```

### 3.Càlcul de la similitud

```
doc1 = normalize(càlcul tf-idf document seleccionat)
for (document in Carpeta != document seleccionat){
    doc2 = normalize(càlcul tf-idf de document)
    similitud = sim(doc1,doc2) //es calcula amb merge per eficiència
    TreeMap.put(similitud, document)
}
return TreeMap
```

## 2.2.2 Estructura de dades utilitzades:

En aquest cas hem utilitzat la mateixa estructura de dades de dues maneres diferents, les taules de Hash (HashMap) de la manera següent:

**HashMap<String, Integer> tf\_hash;**

Key: Type

Value: Freqüència de la paraula en el document

**HashMap<String,Document> posting\_list;**

Key: Type

Value: Documents en els qual apareix la clau

## 2.3 Algorisme cerca per Booleans

No es tracta d'un algorisme en si, però sí que hi ha maneres més eficients i menys eficients de dur a terme aquest cerca. Aprofitant el fet de tenir tots els types del conjunt de documents i els documents en els quals apareixen aquests (Inverted File) fem la cerca de la manera més eficient que buscar per cada contingut, en un vector... tal i com veurem a continuació. A continuació es pot veure com ha estat realitzada aquesta cerca.

### 2.3.1 Realització:

```
for(cada llistat de paraules entre AND, OR, !, {} de l'expressió){
    posting_list = Inverted File calculat prèviament al carregar els documents
    set<Document> aux = buit
    for(paraula en cada llistat de paraules entre AND, OR, !, {}){
        if(paraula existeix a la posting_list){
            afegir al set<Document> docs tots els documents que es troben
            com a valor
        }
    }
    if (OR){
        aux = aux u docs    //unió
    }
    else{ //AND
        aux = aux n docs    //intersecció
    }
}
```

Amb paraules seria el següent:

1. Gràcies a l'Inverted File, un HashMap construït a partir de: (paraula, llista de documents d'aquella paraula) podem fer, per cada llistat de paraules entre AND, OR, !, {}, realitzem una cerca d'una paraula "p" en l'Inverted File i retornar un set de documents en els quals es troba aquella paraula.

2. Segons aquest set i l'organització de l'expressió booleana donada, duem a terme una unió del set de documents actual amb l'anterior, en cas de ser una OR, o una resta de la intersecció de documents del set de documents actual amb l'anterior, en cas de ser una AND.
3. Realitzem un altre cop el mateix fins a obtenir un set de documents final en el qual compleix tota l'expressió.

### 2.3.2 Estructures de dades utilitzades:

En aquest cas hem utilitzat dues estructures de dades, les taules de Hash (HashMap) i un vector de la classe Sting:

**String[] expressió;**

**HashMap<String,Document> posting\_list;**

Key: Type

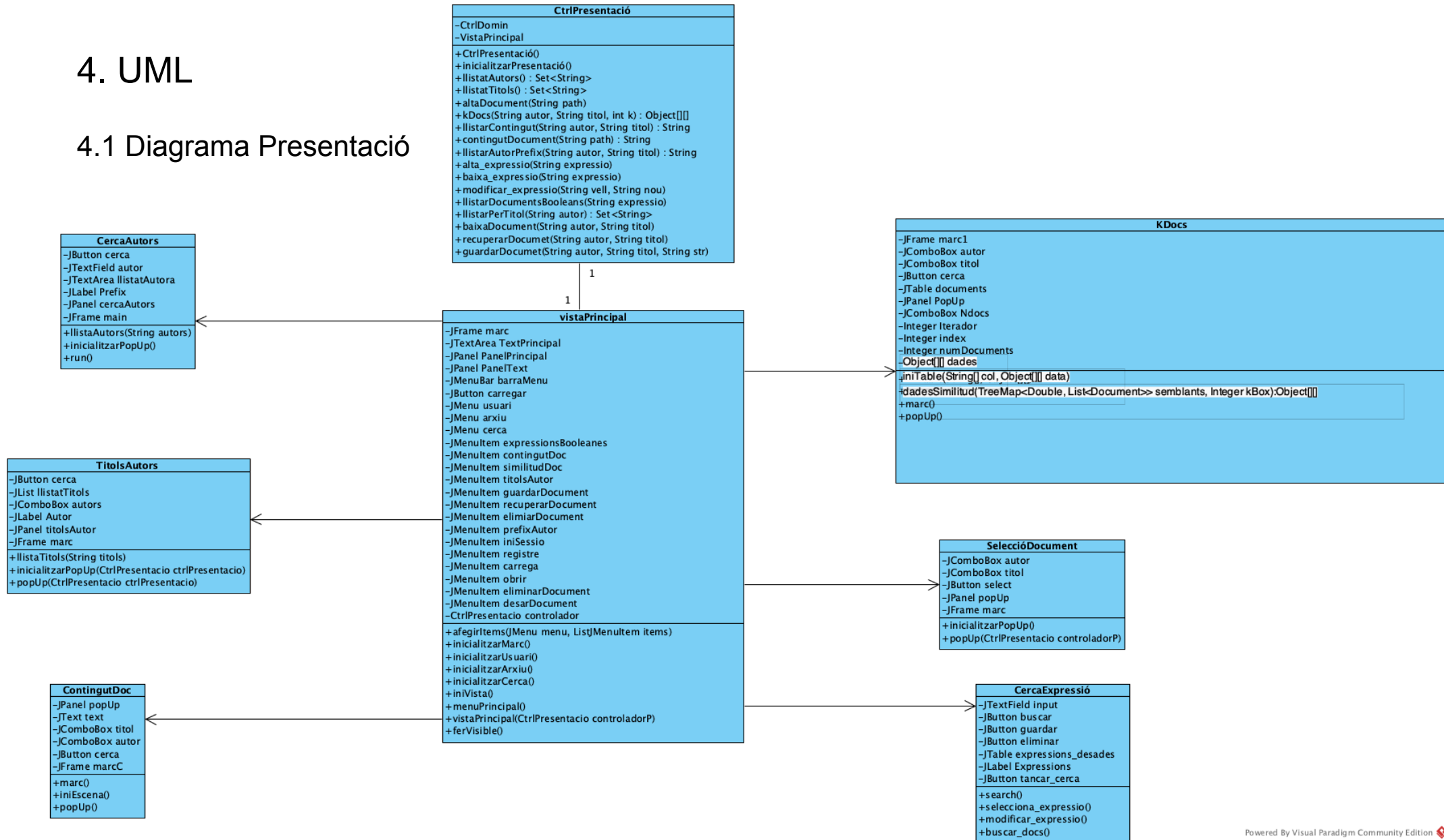
Value: Documents en els qual apareix la clau

### 3. Repartiment de classes

- Classe Document: Lluc
- Classe Algoritme: Lluc
- Classe Expressions: Nora
- Classe ExpressioBooleana: Nora
- Classe Carpeta: Richard, Ruben i Lluc
- Classe ControladorDomini: Richard, Ruben i Lluc
- Classe Main: Richard, Ruben i Lluc
- Capa Presentació: Lluc i Nora
- Capa Persistència: Ruben i Richard

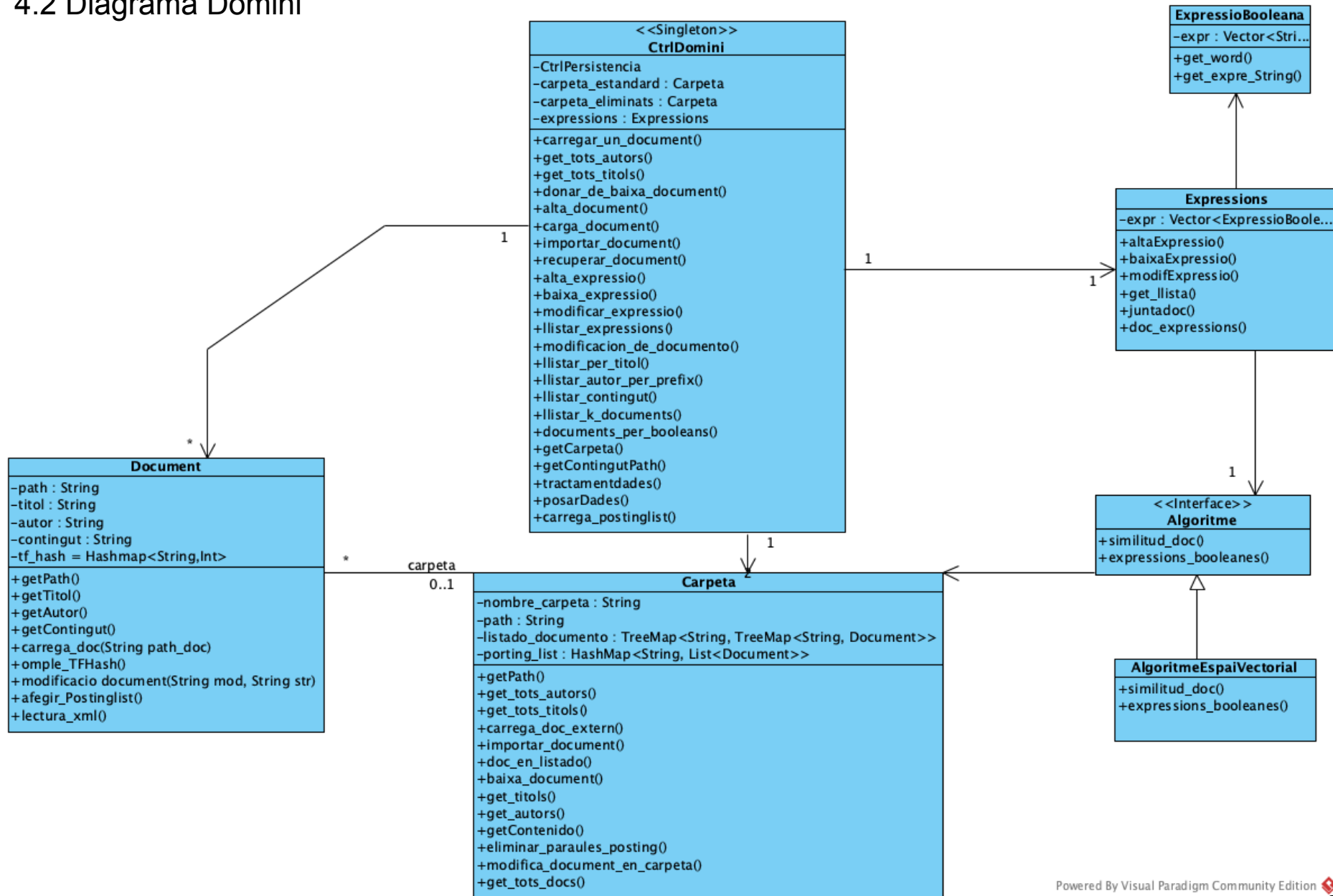
## 4. UML

### 4.1 Diagrama Presentació





## 4.2 Diagrama Domini



## 4.3 Diagrama Persistència

