Gestor de documents

Segona entrega de Projecte de Programació

Q1 2022-23 | Subgrup 31.2

Pere Carrillo

Pol Garrido

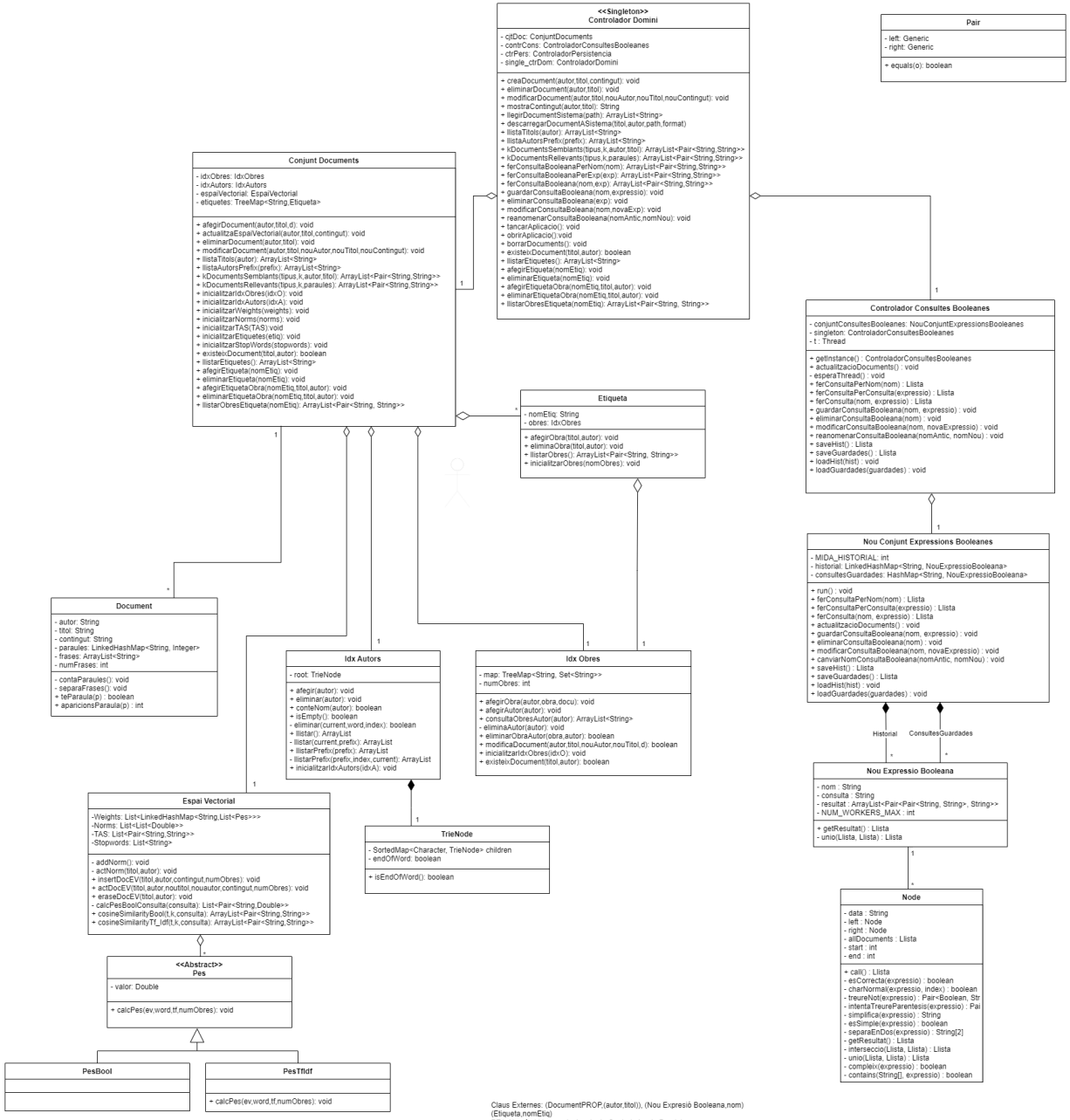
Marc Ordóñez

Laura Pérez

Índex

1 Model capa de domini

1.1 Diagrama



Claus Externes: (DocumentPROP,(autor,titol)), (Nou Expresió Booleana,nom)
(Etiqueta,nomEtiq)
RT1: Només hi ha una instancia de Controlador de Domini

1.2 Descripció

1.2.1 Espai Vectorial

Aquesta classe ens permet emmagatzemar la informació necessària per poder resoldre les consultes de "*k* documents similars respecte a un document D" i "*k* documents similars donat un conjunt de paraules". La informació necessària per aquestes consultes són els pesos de les paraules de tots els documents registrats, amb alguna manera d'identificar cada document. D'això se'n carreguen *Weights* i TAS, essent *Weights* els pesos (amb estrategia booleana i amb tf-idf) i TAS la manera d'identificar conjunt de pesos ràpidament per títol i autor. Com per fer aquestes consultes necessitem computar la similaritat del cosinus, també es guarda la norma dels diferents pesos a *Norms* per no haver de calcular-los a cada consulta. Finalment a l'atribut *Stopwords* tenim totes les *stopwords* en català, castella i angles per no comptar-les a l'espai vectorial.

- **List<String> Stopwords;**
    - Aquí guardem la llista *d'stopwords* en català, castellà i anglès, per computar-les a l'espai vectorial, d'aquesta manera no hem de llegir el fitxer *d'stopwords* cada vegada que el necessitem.
- **List<LinkedHashMap<String,List<Pes>>> Weights;**
    - Ara l'estructura permet emmagatzemar tants tipus de pesos com vulguem.
- **List<List<Double>> Norms;**
    - Igual que a la primera entrega, només que s'ha modificar una mica l'estructura perquè pugui emmagatzemar la norma per cada tipus de pes que es vulgui implementar.
- **List<Pair<String,String>> TAS;**
    - Aquesta estructura ens permet identificar a quin document pertanyen els pesos.

Les funcions són, descomptant *getters* i *setters*:

- **addNorm(): void**
    - Aquesta funció s'encarrega de calcular i afegir a les estructures de dades la norma dels vectors de pesos d'un document nou a afegir.
    - Cost: O(n*m), sent n el nombre de paraules del document i m els tipus diferents de pesos.

- **actNorm(titol,autor): void**
    - Aquesta funció s'encarrega de recalcular la norma dels vectors de pesos d'un document ja registrat al sistema
    - Cost: O(n*m), sent n el nombre de paraules del document i m els tipus diferents de pesos. Actualment hi ha dos tipus de pesos per tant seria O(n)

- **insertDocEV(titol,autor,contingut,numObres): void**
    - Aquesta funció s'encarrega d'afegir un nou document a l'espai vectorial, afegint els pesos i calculant les seves normes. A més comprova si ja existeix un document amb el mateix nom i llença una excepció si és el cas
    - Cost: O(n*m), sent n el nombre de paraules del document i m els tipus diferents de pesos, degut a que es crida a addNorm. Actualment hi ha dos tipus de pesos per tant seria O(n)

- **actDocEV(titol,autor,noutitol,nouautor,contingut,numObres): void**
    - Aquesta funció s'encarrega d'actualitzar un fitxer ja registrat a l'espai vectorial en cas que es produeixi una modificació del mateix. Si no existeix un document prèviament afegit amb aquests identificadors (títol,autor) llença una excepció. En cas contrari, actualitza adientment el document, actualitzant pesos i normes.
    - Cost: O(n*m), sent n el nombre de paraules del document i m els tipus diferents de pesos, degut a que es crida a addNorm. Actualment hi ha dos tipus de pesos per tant seria O(n)

- **eraseDocEV(titol,autor): void**
    - Aquesta funció s'encarrega d'eliminar tota la informació d'un document de l'espai vectorial quan aquest deixa de formar part del sistema. Si no existeix un document amb identificador (titol,autor) llença una excepció.
    - Cost: O(1), ja que es suma de costos constants d'eliminar objectes.

- **calcPesBoolConsulta(consulta): List<Pair<String,Double>>**
    - Aquesta funció s'encarrega de transformar una consulta, que pot ser tot el contingut d'un document o un conjunt de paraules, en pesos seguint l'estratègia booleana (tots a 1 si no són *stopwords*) per poder realitzar les consultes.
    - Cost: O(n), sent n el nombre de paraules de la consulta.

- **cosineSimilarityBool(t,k,consulta): ArrayList<Pair<String,String>>**
    - Aquesta funció s'encarrega de calcular la similaritat del cosinus, utilitzant les dades emmagatzemades a l'espai vectorial, per tal de decidir els k documents més similars respecte la consulta i retornar-los com a resposta a la consulta de "k similars". El paràmetre t s'utilitza per discernir

si la consulta es tracta d'un document o d'un conjunt de paraules. Com el nom indica, utilitza l'estrategia booleana per resoldre les consultes.
    - Cost: O(n*m), sent n el nombre de documents registrats i m el nombre de paraules de la consulta.

- **cosineSimilarityTf_Idf(t,k,consulta): ArrayList<Pair<String,String>>**
    - Aquesta funció s'encarrega de calcular la similaritat del cosinus com l'anterior, però utilitza l'estratègia tf-idf d'assignació de pesos per resoldre les consultes.
    - Cost: O(n*m), sent n el nombre de documents registrats i m el nombre de paraules de la consulta.

1.2.2 Pes / PesBool / PesTfIdf

Aquestes classes ens permeten representar els diversos tipus de pesos amb opció a afegir més amb facilitat, gràcies a l'herència de la clase "Pes". Sempre que es vulgui afegir un nou tipus de pes, només cal heredar de "Pes" i reimplementar la funció "calcPes" amb el mètode que vulguem.

Comentem el mètode:

- **calcPes(ev,word,tf,numObres): void**
    - Aquesta funció s'encarrega de calcular el pes d'una paraula (word) amb la metodologia que es vulgui. En el cas de "PesBool" no es reimplementa, ja que el càlcul del pes amb aquesta metodologia no és res és que asignar 1 si la paraula existeix. En el cas de TfIdf el mètode calcula l'idf i obté el tf com a paràmetre de la funció, ja que és un valor que emmagatzemen als documents. Una vegada calculat el pes de la paraula l'assigna al atribut valor, de la superclasse.
    - Cost: O(n), (en el cas de tfidf) sent n el nombre de documents registrats, en el cas de bool és O(1).

1.2.3 Índex Obres

En aquesta classe guardem la informació necessària per tenir constància dels documents que tenim actualment penjats al sistema. Això ens permet no intentar obrir documents que no existeixin, saber el nombre d'autors que hi ha actualment al sistema, saber el nombre d'obres de cada autor i el títol d'aquestes. Així evitem que, per exemple, si l'usuari afegeix per error algun document al directori on el programa va emmagatzemant la informació de la sessió, s'ignorarà aquest document perquè no formarà part d'idxObres. Aquesta classe també s'utilitzarà a la classe Etiqueta per emmagatzemar les obres que tenen una certa etiqueta.

Els seus atributs son:

- **TreeMap<String, Set<String>> map**
    - Ens permet guardar els documents identificats per títol i autor. Al ser un TreeMap guarda els valors creixentment, de manera que quan vulguem llistar o realitzar alguna consulta se'ns retornaran els valors en ordre.
- **int numObres**
    - Nombre d'obres que tenim a l'índex. Ens serveix per fer càlculs a l'espai vectorial.

Les funcions són, descomptant *getters* i *setters*:

- **afegirObra(String autor, String obra): void**
    - Donat un autor i una obra s'afegeix al TreeMap. Ara ja no es passa DocumentPROP com a paràmetre perquè no cal emmagatzemar-lo a idxObres.
    - Cost: O(log n + log m), sent n el nombre d'autors i m el nombre d'obres de l'autor.
- **afegirAutor(String autor): void**
    - Afegeix un nou autor que no té cap obra.
    - Cost: O(n), sent n el nombre d'autors.

- **consultaObresAutor(String autor): ArrayList<String>**
    - Retorna un ArrayList de Strings amb tots els títols dels documents d'un autor ordenades alfabèticament.
    - Cost: O(log n + m), sent n el nombre d'autors i m el nombre d'obres de l'autor. (log n de trobar l'autor i n de recorregut lineal per llistar).
- **eliminarAutor(String autor): void**
    - S'elimina a l'autor de l'estructura de dades. Aquesta funció només es cridarà quan eliminem una obra d'aquest autor i ja no tingui cap obra al sistema. Només en aquest cas es cridarà a la funció. Per tant, ens assegurem de no estar eliminant a un autor que encara tingués obres. La funció és privada per aquest motiu.
    - Cost: O(log n), sent n el nombre d'autors.
- **eliminarObraAutor(String obra, String autor): boolean**
    - Elimina l'obra d'un autor i retorna true si aquell autor ja no té cap obra. En aquest cas també s'eliminarà a l'autor. En cas que no existeixi l'autor o que no tingui una obra amb el títol indicat saltarà l'excepció i no s'efectuaran els canvis.
    - Cost: O(log n + log m), sent n el nombre d'autors i m el nombre d'obres de l'autor.
- **getTitolsAutors(): ArrayList<Pair<String, String>>**
    - Retorna un ArrayList de Pairs amb totes les obres d'un autor. El primer element del pair és el títol i el segon l'autor.
    - Cost: O(log n + log m), sent n el nombre d'autors i m el nombre d'obres de l'autor.
- **modificaDocument(String autor, String titol, String nouAutor, String nouTitol): boolean**
    - S'adapta l'estructura d'índex obres per actualitzar la informació amb els nous títols i autors. És a dir, si tenim un document al que s'ha canviat el nom de l'autor o del títol, s'haurà d'eliminar l'obra de l'autor original i afegir l'obra al nou autor o canviar el nom de l'obra en cas que es mantingui l'autor. Si a l'eliminar l'obra antiga l'autor antic es queda sense

obres, s'eliminarà a l'autor antic (ja no té obres a l'aplicació) i es retornarà true. En cas contrari es retorna false.
    - Retornem true i false per indicar a conjunt documents si ha d'eliminar a l'autor d'índex autors.
    - Cost: O(log n + log m), sent n el nombre d'autors i m el nombre d'obres de l'autor. En cas pitjor és el cos d'eliminar obra més el d'afegir obra i tots dos són O(log n + log m).
- **getNumAutors(): int**
    - Retorna el nombre d'autors de l'estructura.
    - Cost: O(1).
- **getNumObres(): int**
    - Retorna el nombre d'obres que es troben al sistema en aquell moment.
    - Cost: O(1).
- **inicialitzarIdxObres(TreeMap<String, Set<String>> idxO): void**
    - Setter de la classe. Es crida a l'iniciar l'aplicació per restaurar la informació de la darrera sessió:

1.2.4 Índex Autors

Aquesta classe és una estructura (Trie) en què s'emmagatzemen els noms dels autors. Aquesta estructura ens permet mantenir els noms en ordre alfabètic i llistar per prefixos.

L'únic atribut que té és:

- **TrieNode root**
    - Un Trie amb els noms dels autors.

Les funcions són, descomptant *getters* i *setters*:

- **IdxAutors()**
    - Constructora. Crea un nou TrieNode que serà l'arrel de l'estructura.
    - Cost: O(1)
- **afegir(String autor): void**
    - Afegeix l'autor a l'índex.
    - Cost: O(m), sent m el nombre de lletres de l'autor (perquè el nombre de fills està limitat al nombre de lletres de l'abecedari).
- **conteNom(String autor): boolean**
    - Retorna cert si l'autor pertany a l'índex i fals si no existeix.
    - Cost: O(m), sent m el nombre de lletres de l'autor
- **isEmpty(): boolean**
    - Retorna cert si no hi ha cap autor i fals en cas contrari.
    - Cost: O(1)
- **eliminar(TrieNode current, String word, int index): boolean**
    - Funció recursiva per eliminar una paraula donat un node, una paraula i un índex per indicar la lletra de la paraula que volem eliminar. Si el node actual conté la lletra de la paraula word que està en la posició index, es mira si conté un fill amb el següent caràcter de la paraula i fa una crida recursiva a aquest node fins que arriba a l'última lletra. Si pel camí arriba a un node que cap dels seus fills no és el següent caràcter de la paraula vol

dir que la paraula no existeix i llençarà l'excepció. Si a l'eliminar la paraula ja no penja cap paraula d'aquell node, s'elimina el node sencer.
    - Cost: O(m), sent m el nombre de lletres de word.
- **llistar(): ArrayList**
    - Retorna un ArrayList de Strings amb tots els autors de l'índex ordenats alfabèticament.
    - Cost: O(n), sent n el nombre d'autors que es troben a l'índex.
- **llistar(TrieNode current, String prefix): ArrayList**
    - Llista tots els autors (ordenats alfabèticament) que es troben a partir d'un node. És a dir, tots els autors que comencen pel prefix que va des de l'arrel fins a aquell node.
    - Cost: O(n), sent n el nombre d'autors que es troben a partir del node current.
- **llistarPrefix(String prefix): ArrayList**
    - Llista tots els autors que comencen pel prefix donat. Es va iterant pels nodes fins arribar a la profunditat de l'última lletra del prefix i a partir d'allà es llisten tots els autors que compleixin el prefix.
    - Cost: O(m+n), sent n el nombre d'autors que comencen pel prefix i m el nombre de caràcters del prefix.
- **llistarPrefix(String prefix, int index, TrieNode current): ArrayList**
    - Aquesta funció serveix per arribar fins al node que conté l'última lletra del prefix. índex indica la posició de la lletra del prefix en la que ens trobem. La funció mirarà si un dels fills d'aquest node és la següent lletra del prefix i cridarà a la funció recursiva amb aquell fill i la següent posició de l'índex. Si no troba cap fill amb la següent lletra del prefix, no hi ha cap autor que comenci per aquell prefix i retorna una llista buida.
    - Cost: Cost: O(m+n), sent n el nombre d'autors que comencen pel prefix i m el nombre de caràcters del prefix.
- **inicialitzarIdxAutors(ArrayList idxA): void**
    - Setter de la classe. Es crida a l'iniciar l'aplicació per restaurar la informació de la darrera sessió.
    - Cost: O(n), essent n el nombre d'elements de idxA.

1.2.5 Docu

Aquesta classe és l'antiga classe DocumentPROP, però vam canviar el nom per a que no es confongui amb document propietari. Emmagatzema els atributs de tots els documents, és a dir, el nom de l'autor, el títol del document i el contingut del document, a partir del qual calculem les paraules i les seves aparicions, les frases i el nombre de frases. Ens serveix per operar amb els documents i obtenir informació requerida per a fer consultes o per calcular funcions de l'espai vectorial.

Té els següents atributs:

- **String autor**
    - El nom de l'autor.
- **String titol**
    - El nom del títol.
- **String contingut**
    - El contingut tal com s'ha introduït.
- **LinkedHashMap<String, Integer> paraules**
    - Les paraules amb el nombre d'aparicions en el contingut, eliminant les que comencen per article + apòstrof.
- **ArrayList<String> frases**
    - Una llista de les frases, eliminant els signes de puntuació.
- **int numFrases**
    - El nombre de frases que hi ha en un document.

També té els següents mètodes:

Cada cop que canviem el contingut es cridaran aquestes dues funcions per assignar-li els valors corresponents als atributs numFrases, paraules i frases.

- **contaParaules(): void**
    - Aquesta funció agafa totes les paraules que estan al contingut, treu els signes de puntuació i conta les seves aparicions. També treu les paraules que comencen amb una lletra i apòstrof.

    - Cost: O(n), n la mida del contingut

- **separaFrases(): void**
    - Separa les frases, tenint en compte que frases són totes les que acaben amb un punt, una exclamació o un interrogant. També elimina les exclamacions i els interrogants invertits.
    - Cost: O(n), n la mida del contingut.

Per consultar les paraules del document, tenim aquestes dues funcions:

- **teParaula(String p): boolean**
    - Retorna cert si la paraula p pertany al document.
    - Cost: O(1).
- **aparicionsParaula(String p): int**
    - La paraula p ha d'existir, retorna el nombre d'aparicions de la paraula.
    - Cost: O(1).

1.2.6 Node

Aquesta classe s'encarrega d'emmagatzemar la informació de cada expressió booleana. Ara es guarden en forma d'arbre, on els vèrtex poden ser operadors (&, |) o literals.

Té els següents atributs:

- **String data**
    - Expressió booleana formada pels operadors & | i !, conjunts de paraules (delimitats per {}), seqüències de paraules (delimitades per “”), o paraules soltes com a operands i parèntesis.
- **Node left, right**
    - Els fills esquerre i dret si en té (Sinó son null).
- **boolean negat**
    - Un booleà que indica si l'expressió està negada o no.
- **static ArrayList<Pair<Pair<String, String>, String>> allDocuments**
    - La llista de tots els documents de l'aplicació. S'actualitza cada cop que s'afegeix o elimina un document.
- **int start, end**
    - La posició d'inici i final dels documents que ha de tractar una instància en concret. Més endavant explicarem com i perquè les utilitzem.

I els següents mètodes:

- **Constructora:**
    - La constructora d'un node comprova que l'expressió (data) sigui correcta, treu els possibles parèntesis i NOTs al principi, separa l'expressió en dos i crea dos Nodes més, el fill dret i l'esquerra (si l'expressió és un literal, els fills tenen valor null). A més se li passen els índexs dels documents inicials i finals que ha de comprovar si existeixen.
- **call() : ArrayList<Pair<Pair<String, String>, String>>**
    - Cada thread creat a **NouExpressioBooleana** crida a aquesta funció, que crida a **getResultat()**, explicada més endavant.

- **simplifica() : String**
    - Aquesta funció intenta eliminar tots els possibles parèntesis o NOTs generals que afectin a tota l'expressió. Per fer-ho utilitza les funcions **intentaTreureParentesis(String)** i **treureNot(String)** que son iguals que a l'entrega 1. Retorna l'expressió simplificada.
- **getResultat() : ArrayList<Pair<Pair<String, String>, String>>**
    - L'única cosa que canvia d'aquesta funció respecte a l'entrega anterior és que ara es mirem només els documents amb índex de **start** a **end**.
    - Aquesta funció és la que implementa la recursivitat de l'algorisme. Té dos casos base: El primer, si l'expressió ja tenia solució retorna, i el segon, si l'expressió és simple, la resol i retorna.

      En cas que no es compleixi cap dels dos casos base separa en dos l'expressió i crida a getResultat() de cada fill.

      Finalment, si havia separat per una OR, fa la unió dels resultats, i si havia separat per una AND, la intersecció. En cas que la consulta estigués negada s'inverteixen les funcions per calcular el resultat.

    - Cost: El cost d'aquesta funció és el mateix que el de ferConsulta(), molt difícil de calcular teòricament.
- **esCorrecta(String expressio) : boolean**
    - Aquesta consulta retorna cert si l'expressió booleana **expressio** és correcta i fals en cas contrari. Per saber si una expressió és correcta comprovem els següents casos:
        - Que tota l'expressió siguin seqüències de literal - operador - literal tenint en compte els parèntesis. (Un literal pot ser una paraula sola, una seqüència de caràcters delimitada per "" o un conjunt de paraules delimitats per {}).
        - Que no quedi cap parèntesi, claudàtor o cometes sense tancar i que no se'n tanquin de més.
        - Que les NOT s'apliquin sobre un literal, sobre un parèntesis o sobre una altra NOT.

    - Aquesta consulta utilitza la funció **charNormal(String, int)**, per comprovar els caràcters que hi ha després d'una NOT
    - Cost: O(n) sent n el nombre de caràcters de la consulta, ja que hem de recórrer tota la consulta caràcter per caràcter en cas pitjor. En el cas que aquest sigui una NOT haurem de comprovar els següents caràcters diferents d'un espai, però com que l'únic cas en que s'itera per tots els elements restants a charNormal() és en el cas que siguin espais, per les següents no s'iterarà (perquè no seran NOTs) i per tant no fa augmentar el cost.
- **esSimple(String) : boolean**
    - Aquesta consulta retorna cert si l'expressió sobre la que es fa és simple, és a dir, conté només un literal (descrit a la funció anterior).
    - Cost: O(n), ja que en cas pitjor haurem de recórrer tot el String.
- **separaEnDos() : String[2]**
    - Aquesta consulta és la que permet aplicar el nostre algorisme de Dividir i Vèncer. Retorna l'expressió booleana com a String sobre la que s'ha cridat dividida en 2. Intenta dividir-la primer en dos per una OR exterior, i si no n'hi ha cap, per una AND. Sempre ens trobarem en algun d'aquests dos casos, ja que si no, o la consulta era incorrecte o la hem resolt just abans, però si no passa res d'això retornem la consulta a resultat[0] i null a resultat[1].
    - Cost: O(n), ja que en cas pitjor farem dos recorreguts pel String, un per buscar ORs i l'altre per buscar ANDs.
- **eliminaParentesisInutils(String) : boolean**
    - Aquesta funció intenta eliminar un possible parèntesis general que afecti a tota la funció. Si ho aconsegueix retorna true, si no false.
    - Cost: O(n), ja que en cas pitjor haurem de recórrer tot el String. Tot i que si la consulta no comença per parèntesi el cost és O(1).
- **treureNot(String) : boolean**
    - Aquesta funció fa el mateix que la anterior però buscant una NOT.
    - Cost: El mateix que la funció anterior.

- **interseccio(ArrayList< Pair< Pair< String, String>, String>> docs, ArrayList< Pair< Pair< String, String>, String>> altres): ArrayList< Pair< Pair< String, String>, String>>**
    - Es fa la intersecció de les dues llistes docs i altres. Retorna una llista de títol, autor, frase, que pertanyen tan a docs com a altres. Per a això, en el cas que docs sigui més gran, anem mirant si cada element de altres pertany a docs. En cas contrari es comprova si els elements de docs pertanyen a altres. Si l'element pertany a les dues llistes, s'afegeix a la llista resultant.
    - Cost: O(n*m), n la mida de docs i m la mida de altres
- **unio(ArrayList< Pair< Pair< String, String>, String>> docs, ArrayList< Pair< Pair< String, String>, String>> altres): ArrayList< Pair< Pair< String, String>, String>>**
    - Es fa la unió de les dues llistes docs i altres. Retorna una llista de tots els documents pertanyents a docs i altres. Per a això, clonem docs al resultat i li afegim tots els elements de altres que no pertanyen a docs.
    - Cost: Cost: O(n*m), n la mida de docs i m la mida de altres
- **compleix(String) : boolean**
    - Retorna cert si la frase que se li passa compleix l'expressió que se li passa. Comprovem els 3 possibles casos d'expressió: Tenir una paraula sola, un conjunt de paraules o una seqüència de paraules. Utilitza la funció **contains(String[], String),** que comprova si un array conté un String (case insensitive).
    - Cost: O(n), sent n el nombre de paraules de la frase

Utilitzem el mateix algoritme que en la resolució de la primera versió per crear l'arbre, però sense resoldre'l: Si l'expressió és simple es deixa com està, si no, es separa en dos pel primer operador adient, s'afegeixen com a fill dret i esquerre i es crida recursivament a la creadora.

L'algorisme per treure les NOT i els parèntesis innecessaris és exactament el mateix que a la versió 1 però ara es fan els dos a la vegada a la funció simplifica.

1.2.7 Nou Expressió Booleana

Aquesta classe s'encarrega de guardar una expressió booleana, el seu nom i el resultat. També gestiona la creació de threads que s'utilitzen per resoldre la consulta i de recuperar tots els resultats parcials un cop han acabat tots.

Aquesta classe guarda una consulta booleana. Una consulta booleana la definim com: La consulta en sí (com a String), el seu nom i el resultat de la consulta, si el sabem.

Hem pres la decisió de només poder buscar caràcters especials (&, |, {, (, !,), }) si els posem entre cometes, ja que si no es produeixen moltes ambigüitats i potser una consulta on l'usuari s'ha equivocat pot seguir donant un resultat.

Té els següents atributs:

- **String nom**
    - El nom de l'expressió. Si no en té és igual a consulta.
- **String consulta**
    - La consulta com a string.
- **ArrayList<Pair<Pair<String, String>, String>> resultat**
    - El resultat de la consulta. És null si no s'ha calculat encara.

I els següents mètodes (a part de getters i setters trivials):

- **getResultat() : ArrayList<Pair<Pair<String, String>, String>>**
    - Tot i ser un getter, aquesta operació no és trivial. Es crida quan es vol obtenir el resultat d'una expressió booleana. Si ja estava calculat abans es retorna el guardat. Si no, es creen un màxim de 10 threads que executen 10 instàncies de Node diferents entre les quals es divideixen tots els documents del sistema i cadascun resol la consulta booleana en aquells. Un cop han acabat, el thread principal recull tots els resultats i retorna la unió de tots ells. Actualment està programat perquè els documents es separin en paquets de 1 o més, per demostrar el funcionament però en una versió final es canviaria a que cada thread se li assignin com a mínim 10 documents ja que amb un sol document per thread (per exemple) es

perd més temps separant i ajuntant els resultats que el que es guanya amb la paral·lelització.

1.2.8 Nou Conjunt Expressions Booleanes

Aquesta classe és l'encarregada de guardar i gestionar les consultes booleanes. Ha de poder rebre una consulta i retornar un resultat, mantenir un historial i permetre guardar un conjunt de consultes a decidir per l'usuari. També ha de permetre guardar aquesta informació a disc al tancar l'aplicació i recuperar-la a l'obrir-la.

Té el mateix que l'antiga ConjuntConsultesBooleanes però amb les noves classes, excepte per l'historial i la llista de consultesGuardades, que ara tenen com a clau el nom de l'expressió.

L'historial el guardem en un LinkedHashMap, ja que ens interessa poder fer ràpidament insercions al principi i eliminacions de l'últim element com en una cua, però podent tenir accés a qualsevol dels elements del mig. La mida màxima de l'historial la guardem en un atribut static i final de la classe. Quan fem una consulta i es supera aquest nombre, s'elimina la consulta feta més antigament i s'afegeix la nova al principi. A l'historial guardem les expressions i els resultats, per poder donar un resultat ràpidament si la consulta ja s'havia fet anteriorment, però per tant, cada cop que s'afegeix, elimina o modifica un document hem recalcular els resultats de totes les consultes.

Per oferir una gestió de les consultes booleanes permetem a l'usuari poder guardar consultes, modificar alguna que ja tenia guardada o eliminar-ne una. El conjunt de consultes de l'usuari les guardem en un HashMap, ja que és una estructura que ens permet poder afegir, eliminar i consultar un element en qualsevol posició ràpidament. Tenim, per això, el mateix problema que amb l'historial, i haurem recalcular totes les consultes cada cop que s'afegeix, modifica o elimina un document.

Aquesta classe volem que la faci servir un thread secundari, per no haver de fer esperar a l'usuari mentre es resolen les consultes booleanes.

- **int MIDA_HISTORIAL**
    - Mida màxima de l'historial.

- **LinkedHashMap<String, NouExpressioBooleana> historial**
    - Guardem les últimes "MIDA_HISTORIAL" consultes fetes per l'usuari, permetent reutilitzar les més recents. Necessitem poder afegir una consulta al principi i eliminar-ne la última de forma ràpida (com una cua), però també accedir a un element aleatori, per tant un LinkedHashMap és la millor opció, amb clau el nom de la consulta i valor la instancia de NouExpressioBooleana. Per mantenir la coherència cada cop que s'afegeix, elimina o modifica un document recalculem totes les expressions booleanes que teniem guardades a l'historial.
- **HashMap<String, NouExpressioBooleana> consultesGuardades**
    - Guardem totes les consultes que l'usuari ha demanat de guardar, en un mapa amb clau el nom de la consulta i valor la instancia de NouExpressioBooleana. Això ens permet consultar, inserir i eliminar de forma ràpida. Per mantenir la coherència cada cop que s'afegeix, elimina o modifica un document recalculem totes les expressions booleanes que teniem guardades.

Ofereix els següents mètodes:

- **run() : void**
    - És el mètode que crida el thread secundari quan es crea. Calcula i guarda el resultat de totes les expressions guardades per l'usuari i a l'historial.
- **ferConsulta(String Nom, String Expressio) : ArrayList<Pair<String, String>>**
    - També existeixen les variants **ferConsultaPerNom(String)** i **ferConsultaPerConsulta(String)** que criden a aquesta funció amb un paràmetre adicional.
    - Aquesta funció actualitza l'historial i crida a **getResultat()** de NouExpressioBooleana.
- **actualitzacioDocuments() : void**
    - Aquesta funció es crida just abans de **run()**, per invalidar tots els resultats que teniem guardats al crear/modificar/eliminar un document.
- **guardarConsultaBooleana(String) : void**

    - Funció que comprova que la consulta sigui correcta i la guarda. En cas que no sigui correcta llança una excepció.
- **modificarConsultaBooleana(String, String) : void**
    - Aquesta funció rep dos paràmetres, el nom de la consulta antiga i la nova expressió. El que fa la funció és eliminar la antiga i crear i guardar la nova, fent saltar excepcions en els casos corresponents.
- **eliminarConsultaBooleana(String) : void**
    - Aquesta funció elimina una consulta de les guardades per l'usuari. En cas que no existeixi fa saltar una excepció.
- **canviarNomConsultaBooleana(String nomAntic, String nomNou):**
    - Aquesta funció rep el nom antic d'una consulta i el canvia per el nom nou.
- **saveHist() : ArrayList<Pair<String, String>>**
    - Retorna les expressions guardades a l'historial per guardar-les a disc.
- **saveGuardades() : ArrayList<Pair<String, String>>**
    - Retorna les expressions guardades per l'usuari per guardar-les a disc.
- **loadHist(ArrayList<Pair<String, String>>) : void**
    - Afegeix totes les expressions de la llista a l'historial. S'utilitza només a l'iniciar l'aplicació
- **loadGuardades(ArrayList<Pair<String, String>>) : void**
    - Afegeix totes les expressions de la llista a les consultes guardades. S'utilitza només a l'iniciar l'aplicació

1.2.9 Controlador Consultes Booleanes

Aquesta classe és un Singleton, és a dir, només pot haver-hi una instancia, i així ens assegurem que tothom treballa sobre la mateixa instancia. És l'encarregada de crear i gestionar el thread secundari per cridar a **NouConjuntExpressionsBooleanes** i fer tota la feina mentre l'usuari pot anar fent altres coses. Com a atributs té la instància de NouConjuntExpressionsBooleanes, ella mateixa (pel singleton) i el Thread que utilitza.

Els únics mètodes importants son:

- **actualitzacioDocuments() : void**
    - Crida a **actualitzacioDocuments()** de **NouConjuntExpressionsBooleanes** que invalida els resultats de totes les consultes i crea un thread perquè les resolgui paral·lelament.
- **esperaThread() : void**
    - Atura l'execució del programa mentre el thread secundari no acaba. Quan acaba retorna.

Tota la resta de funcions criden primer a **esperaThread()** i després a una funció amb el mateix nom a NouConjuntExpressionsBooleanes. Això ho fem per evitar problemes de sincronització, i assegurar-nos que el thread secundari hagi acabat la feina abans de retornar res.

1.2.10 Conjunt Documents

Aquesta classe representa el conjunt de tots els documents, ja que manté els índexs dintre seu. Actua com l'element comunicador entre els documents emmagatzemats i l'espai vectorial o el conjunt de consultes booleanes, entre d'altres funcions. És una de les classes que més canvis a patit respecte la primera entrega, ja que farà de nexe d'unió entre les capes de presentació i persistència. Hem hagut d'afegir les funcions per carregar a l'aplicació i guardar a sistema la informació de les classes per mantenir informació entre sessions. També hem reimplementat funcions ja existents per fer que quan vulguin obtenir informació d'un document no la busquin a idxObres (que és on s'emmagatzemava abans la informació del document) sinó a persistència.

Els seus atributs son:

- **int numDocs**
    - Nombre de documents que tenim al nostre conjunt de documents.
- **IdxObres idxObres**
    - Tots els documents guardats identificats pel títol i l'autor.
- **IdxAutors idxAutors**
    - Els autors que tenim al nostre sistema.
- **EspaiVectorial espaiVectorial**
    - Els documents guardats amb pesos segons les paraules.
- **TreeMap<String, Etiqueta> etiquetes**
    - Estructura per emmagatzemar totes les etiquetes que haguem creat.

Detallem ara les funcions de la classe:

- **afegirDocument(autor,titol,d): void**
    - Afegeix un document als indexs donat un autor i un títol. La funció d'actualitzar l'espai vectorial l'hem separat en una funció a part perquè és una funció que ha d'accedir als documents del sistema. Com que el nou

document no es guarda a sistema fins que no s'acaba la crida a aquesta funció, si intentem fer alguna operació que consulti les obres del sistema tindrem una inconsistència entre la informació dels índexs (en els que ja apareix el nou document i la capa de persistència on encara no s'ha guardat el nou document).
    - Cost: O(log n + log m), sent n el nombre d'autors i m el nombre d'obres d'aquest.
- **actualitzarEspaiVectorial(String autor, String titol, String contingut) : void**
    - És la segona part d'afegir un document. Un cop s'ha guardat a memòria ja es pot fer les operacions pertinents a espai vectorial. Per fer-ho creem un Docu per obtenir les paraules del document a partir d'autor, titol i contingut.
    - Cost: O(log n + log m), sent n el nombre d'autors i m el nombre d'obres d'aquest.
- **eliminarDocument(autor,titol) : void**
    - Elimina un document donat el seu titol i autor de les estructures de dades. Si l'obra no existeix, no s'elimina res. Si a l'eliminar l'obra a idxObres aquell autor es queda sense obres, es crida a idxAutors per eliminar a l'autor (ja no té cap obra i considerem que deixa de formar part del sistema). Ara també s'eliminarà el document de totes aquelles etiquetes que tingués.
- **modificarDocument(String autor, String titol, String nouAutor, String nouTitol, String nouContingut) : void**
    - S'actualitza idxObres per modificar l'obra antiga amb la nova informació. Si l'autor antic es queda sense obres se l'elimina d'idxAutors. També s'actualitza l'espai vectorial.
- **llistaTitols(autor): ArrayList<String>**
    - Retorna la llista de títols d'un autor
- **llistaAutorsPrefix(prefix): ArrayList<String>**
    - Retorna la llista d'autors per prefix

- **kDocumentsSemblants(tipus,k,autor,titol): ArrayList<Pair<String,String>>**
    - Retorna els k documents més semblants al consultat amb l'estratègia d'assignació de pesos escollida amb el paràmetre tipus
- **kDocumentsRellevants(tipus,k,paraules): ArrayList<Pair<String,String>>**
    - Retorna els k documents més rellevants per la consulta en forma de conjunt de paraules amb l'estratègia d'assignació de pesos escollida amb el paràmetre tipus
- **getAllTitolsAutors() : ArrayList<Pair<String,String>>**
    - Retorna una llista amb totes les obres en forma de Pair <títol, autor>. Aquesta llista per defecte ve ordenada alfabèticament per autor i títol. Primer criteri d'ordenació són els autors i el segon són els títols, tenint en comptes que les lletres majúscules van abans que les minúscules.
- **inicialitzarEtiquetes(ArrayList<Pair<String, TreeMap<String, Set<String>>>> etiq) : void**
    - Es rep com a paràmetre un ArrayList de Pairs de *<nomEtiqueta, obresQueTenenAquestaEtiqueta>*. El nom és un String i les obres que tenen aquella etiqueta es guarden en forma d'arbre amb key nom de l'autor i value totes les obres d'aquell autor que tenen l'etiqueta en forma de set. El que farà la funció és recórrer l'ArrayList creant una nova etiqueta per a cada element de la llista, i inicialitzant aquella etiqueta amb el TreeMap dels autors i les obres que tenen l'etiqueta. Cadascuna d'aquestes obres s'afegirà al TreeMap de conjunt documents amb key nom de l'etiqueta i value l'etiqueta que acabem de crear.
- **existeixDocument(String titol, String autor) : boolean**
    - Retorna cert si a idxObres hi ha un document amb el títol i autor que passem com a paràmetres i retorna fals en cas contrari.
- **llistarEtiquetes() : ArrayList<String>**
    - Recórrer tot el TreeMap de les etiquetes i retorna un ArrayList de Strings amb els noms de les etiquetes (les key dels elements del map).
- **afegirEtiqueta(String nomEtiq) : void**
    - Si no existeix una etiqueta amb aquest nom, crea una nova etiqueta amb el nom que rep com a paràmetre i l'afegeix al TreeMap.

- **eliminarEtiqueta(String nomEtiq) : void**
    - Si existeix una etiqueta amb el nom que rep com a paràmetre, l'elimina del TreeMap. Altrament no l'elimina perquè directament no existia.
- **afegirEtiquetaObra(String nomEtiq, String titol, String autor) : void**
    - Si existeix una etiqueta amb el nom que rep com a paràmetre, afegeix a l'etiqueta associada aquest nom un document identificat per títol i autor.
- **eliminarEtiquetaObra(String nomEtiq, String titol, String autor) : void**
    - Si existeix una etiqueta amb el nom que rep com a paràmetre, elimina al document identificat per títol i autor de l'etiqueta.
- **llistarObresEtiqueta(String nomEtiq) : ArrayList<Pair<String, String>>**
    - LLista les obres de l'etiqueta que té el nom que rep com a paràmetre. Si l'etiqueta no existeix, es retorna un ArrayList buit. En cas contrari, es retorna un ArrayList de Pairs títol i autor per a cadascun dels documents que tinguin aquella etiqueta.
- **canviarNom(String nomOriginal, String nouNom) : void**
    - Si l'etiqueta original existeix, si li canvia el nom pel nouNom (ja haurem comprovat que no sigui un nom que estigui capat pel sistema).

També té molts mètodes del tipus inicialitzar*NomEsctructura*, que l'únic que fan és cridar a funcions d'altres classes amb el mateix nom. Es fan servir quan volem restaurar els valors d'una sessió o quan volem resetejar la sessió i eliminar tots els documents que hi teníem carregats (i conseqüentment la informació dels índexs o d'altres classes involucrades).

1.2.11 Etiqueta

La classe etiqueta ens permet dividir els documents per categories per facilitar la gestió a l'usuari i permetre filtrar els documents que se'ns mostren per pantalla. Per fer-ho, s'assigna una etiqueta (o varies) al document, la qual té un nom identificatiu, i una vegada està assignat, quan l'usuari demani veure només les obres d'una categoria, el sistema serà capaç de llistar-li les obres que tinguin la corresponent etiqueta. Evidentment, l'usuari també podrà eliminar etiquetes o canviar el nom a les ja creades.

Permetrem llistar per una o vàries etiquetes i sempre tindrem l'opció de mostrar tots els documents (independentment de si tenen o no etiquetes) o mostrar els Favorits. L'etiqueta Favorits es crearà per defecte i no se li podrà modificar el nom. S'utilitzarà per indicar els documents que l'usuari vulgui tenir més a mà i es mostraran a la pantalla inicial del programa per facilitar-ne l'accés.

Els atributs són:

- **String nomEtiq**
    - Nom de l'etiqueta.
- **IdxObres obres**
    - Reutilitzem la classe índex obres que servia per guardar títols i autors i la utilitzem per guardar els documents que tinguin l'etiqueta.

Les funcions d'aquesta classe són:

- **afegirObra(String titol, String autor) : void**
    - S'afegeix una obra identificada per títol i autor al conjunt d'obres de l'etiqueta.
- **eliminarObra(String titol, String autor) : void**
    - S'elimina una obra identificada per títol i autor del conjunt d'obres de l'etiqueta.
- **llistarObres() : ArrayList<Pair<String, String>>**

    - Es retorna un ArrayList de Pair títol i autor ordenades alfabèticament amb tots els documents que tenen aquella etiqueta.
- **getNom() : String**
    - Retorna el nom de l'etiqueta.
- **inicialitzarObres(TreeMap<String, Set<String>> nomObres) : void**
    - Se li passa com a paràmetre un TreeMap amb key autor i un set amb els noms de les obres d'aquell autor. Per tant, se li passen tots els autors i les obres de cadascun dels autors. La funció farà que els documents que tenen l'etiqueta siguin únicament els que ha rebut en el TreeMap nomObres.
- **getObres() : TreeMap<String, Set<String>>**
    - Retorna un TreeMap amb la mateixa estructura que la funció anterior. És a dir, un TreeMap d'autors amb un set de les seves obres que tinguin l'etiqueta.
- **canviarNom(String nomOriginal, String nouNom) : void**
    - Es canvia el nom de l'etiqueta excepte si l'etiqueta és la Favorits, que no se li pot canviar el nom. Tampoc es permet posar a una etiqueta el nom "Tots", ja que és una paraula reservada pel sistema per indicar que volem visualitzar tots els documents del programa (independentment de les seves etiquetes). Si permetéssim l'etiqueta Tots a la llista d'etiquetes que podem utilitzar per llistar apareixeria dos cops Tots i podria portar a confusió. Aquesta comprovació de que el nou nom no sigui ni Tots ni Favorits ja s'haurà fet anteriorment.

1.2.12 Controlador Domini

Aquesta classe actuara com element de comunicació entre la capa de presentació i la de domini, o entre la de persistencia i la de domini. Es la classe on es reben les consultes i es deriven a les classes especialitzades.

Els atributs d'aquesta classe són:

- **ConjuntDocuments cjtDoc**
    - Aquí és on s'instància la clase conjunt documents
- **ControladorConsultesBooleanes contrCons**
    - La instància de Controlador Consultes Booleanes
- **ControladorPersistencia ctrPers**
    - La instància de Controlador Persistència
- **ControladorDomini single_ctrDom = null**

Passem a comentar els mètodes implementats a aquesta classe:

- **creaDocument(autor,titol,contingut) : void**
    - Crea un document comprovant que el títol o el autor no siguin buits. Afegeix el document a conjuntDocuments, i actualitza les estructures de dades perquè les consultes posteriors es puguin resoldre correctament.
    - Cost: O(n), sent n el nombre de paraules del document, degut al cost del càlcul del idf.
- **modificarDocument(autor,titol,nouAutor,nouTitol,nouContingut) : void**
    - Modifica un document crida a que s'actualitzi a les diferents estructures de dades (indexs, controlador consultes i espai vectorial) i elimina el document antic de memòria i guarda el nou document a memòria. No es fa distinció entre si s'ha modificat únicament el contingut o si ha canviat el títol i l'autor. En tots els casos s'ha de modificar el fitxer amb la informació del document. Per tant, sempre l'eliminem i el tornem a crear amb la informació correcta. A l'hora de modificar el document a conjunt documents (ídxObres) sí que es té en compte què és el que s'ha modificat per no fer feina innecessària. L'últim pas que es fa és el de fer els canvis

pertinents a memòria principal un cop ens hem assegurat que no hi ha cap problema amb títols repetits o altres problemes que poguessin sorgir i que a l'haver-se tractat abans farien saltar una excepció i no s'arribaria a fer la modificació a persistència. En cas que surgís algun problema i no es pogués efectuar la modificació a memòria es faria un catch de l'excepció per restaurar l'estat previ dels índexs i d'altres estructures que ja s'haguessin modificat.

    - Cost: O(n), sent n el nombre de paraules del document, degut al cost del càlcul del idf.
- **eliminaDocument(String autor, String titol) : void**
    - Elimina el document de les estructures de dades que tinguessin constància d'aquest document (idxObres, idxAutors, Consultes Booleanes i Espai Vectorial). També s'actualitza consultes booleanes per què no tingui en compte el document que acabem d'eliminar.
- **mostraContingut(autor,titol) : String**
    - Es crida a la capa de persistència per obtenir la informació d'un document donat un títol i un autor. Retorna un ArrayList de tres elements, on el tercer element és el contingut del document. Aquest contingut serà el que retornem.
- **llegirDocumentSistema(path) : ArrayList<String>**
    - Es crida a la cap de persistència per obtenir la informació d'un document donat un títol i un autor. En aquest cas no es retorna únicament el contingut sinó un ArrayList de tres elements amb títol, autor i contingut.
- **descarregarDocumentASistema(titol,autor,path,format) : void**
    - Crida a persistència per guardar el document identificat per títol i autor (en cas que existeixi) al path en el format especificat (que només podrà ser txt, xml o kitty).
- **llistaTitols(autor) : ArrayList<String>**
    - Llista els títols d'un autor en un ArrayList.
- **llistaAutorsPrefix(prefix) : ArrayList<String>**
    - Llista autors que comencen pel prefix passat com a paràmetre de la funció

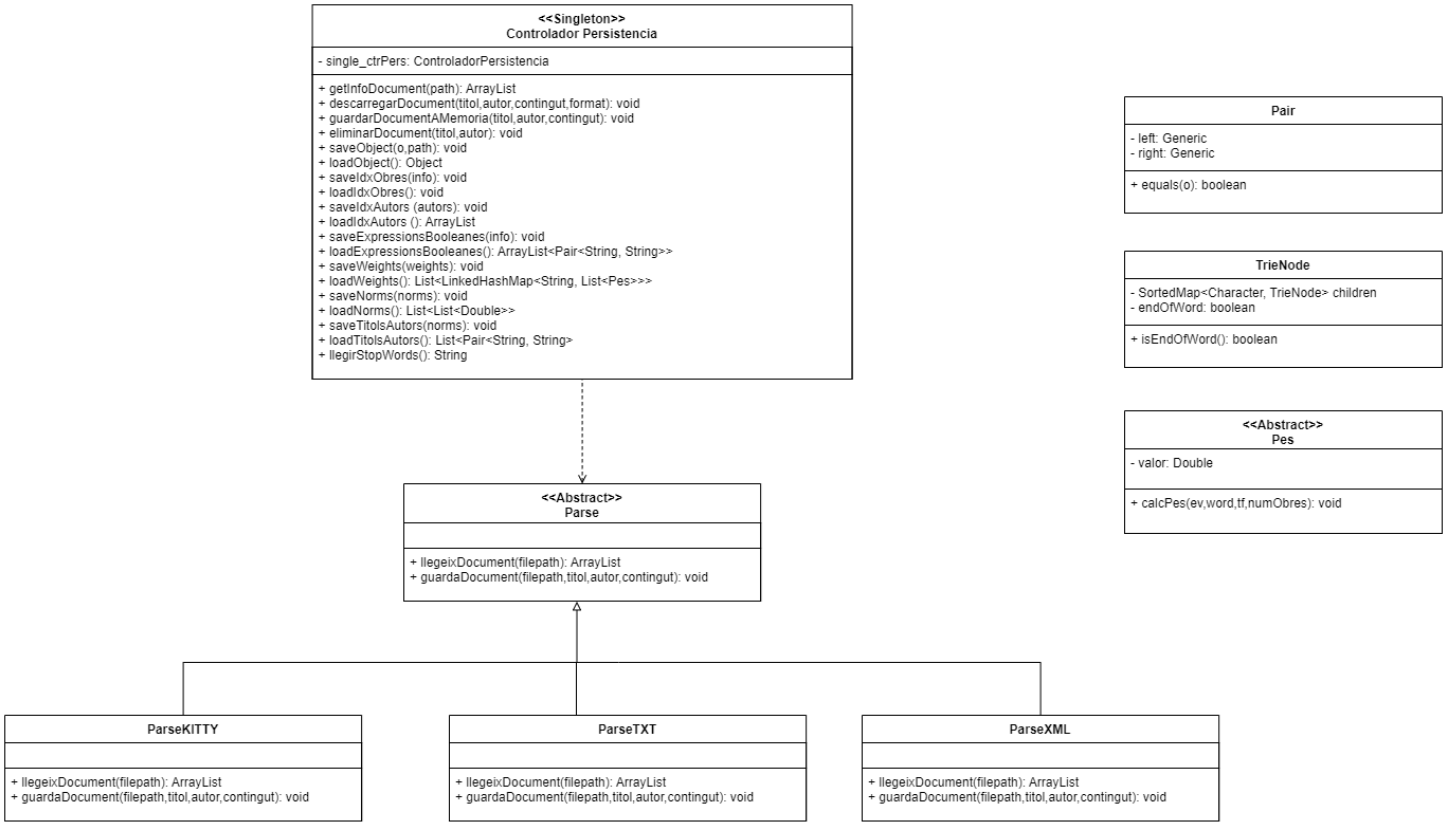
- **kDocumentsSemblants(tipus,k,autor,titol) : ArrayList<Pair<String,String>>**
    - Retorna llista de k parelles títol i autor semblants al document identificat per *autor* i *títol*. Ho retorna en forma d'ArrayList de Pairs títol, autor.
- **kDocumentsRellevants(tipus,k,paraules) : ArrayList<Pair<String,String>>**
    - Retorna llista de k parelles títol i autor rellevants pel conjunt de paraules consultat. Ho retorna en forma d'ArrayList de Pairs títol, autor.
- **ferConsultaBooleanaPerNom(nom) : ArrayList<Pair<String,String>>**
    - Retorna llista de parelles títol i autor que compleixen l'expressió booleana identificada pel seu nom.
- **ferConsultaBooleanaPerExp(exp) : ArrayList<Pair<String,String>>**
    - Retorna llista de parelles títol i autor que compleixen l'expressió booleana passada directament com a paràmetre.
- **ferConsultaBooleana(nom,exp) : ArrayList<Pair<String,String>>**
    - Retorna llista de parelles títol i autor que compleixen l'expressió booleana identificada pel seu nom i exp.
- **guardarConsultaBooleana(nom,expressio) : void**
    - Guarda la consulta booleana amb el nom indicat al sistema.
- **eliminarConsultaBooleana(exp) : void**
    - Elimina la consulta amb expressió *exp*
- **modificarConsultaBoleana(nom,novaExp) : void**
    - Modifica una expressió ja guardada i en modifica l'expressió booleana
- **reanomenarConsultaBooleana(nomAntic,nomNou) : void**
    - Modifica el nom d'una expressió guardada el sistema, canviat-li el nom per un de nou passat com a paràmetre
- **tancarAplicacio() : void**
    - Tanca l'aplicació guardant abans a disc els índexs, la informació de l'espai vectorial, i les expressions usades o guardades explícitament amb un nom identificatiu. També es guarden les etiquetes dels documents. Primerament obté tota la informació a guardar cridant a conjunt documents o a controlador consultes booleanes i després crida a persistència i va guardant un per un aquesta informació.
- **obrirAplicacio() :void**

    - Carrega els índexs de disc per començar la sessió. També haurà de carregar la informació d'espai vectorial, la d'espai vectorial i la d'etiquetes. Actualment això es fa directament a la constructora de controlador domini però creiem que és millor tenir-ho en una funció a part. Només caldrà assegurar-se que la capa de presentació cridi a aquesta funció just a l'obrir el programa (després de crear la instància de controlador domini)
- **borrarDocuments() : void**
    - Fa un reset de tota la informació emmagatzemada. Això ens és útil com a mínim durant les proves per poder borrar tota la informació de cop i fer que no hi hagi cap inconsistència entre les capes ni entre les classes. Per exemple que consultes booleanes no es pensi que existeix un document que segons idxObres no existeix.
- **existeixDocument(titol,autor) : boolean**
    - Comprova si un document existeix.
- **llistarEtiquetes() : ArrayList<String>**
    - Llista el nom de les etiquetes definides al sistema.
- **afegirEtiqueta(nomEtiq) : void**
    - Afegeix una etiqueta al sistema.
- **eliminarEtiqueta(nomEtiq) : void**
    - Elimina una etiqueta del sistema. Si l'etiqueta no existia no farà res.
- **afegirEtiquetaObra(nomEtiq,titol,autor) : void**
    - Assigna una etiqueta a un document.
- **eliminarEtiquetaObra(nomEtiq,titol,autor) : void**
    - Desassigna l'etiqueta amb nom *nomEtiq* de l'obra indicada.
- **llistarObresEtiqueta(nomEtiq) : ArrayList<Pair<String, String>>**
    - Llista les obres que tenen assignada l'etiqueta passada com a paràmetre. Retorna un ArrayList de Pair títol, autor.
- **canviarNom(String nomOriginal, String nouNom) : void**
    - Es canvia el nom de l'etiqueta excepte si l'etiqueta és la Favorits, que no se li pot canviar el nom. Tampoc es permet posar a una etiqueta el nom "Tots", ja que és una paraula reservada pel sistema per indicar que volem visualitzar tots els documents del programa (independentment de les

seves etiquetes). Si permetéssim l'etiqueta Tots a la llista d'etiquetes que podem utilitzar per llistar apareixeria dos cops Tots i podria portar a confusió.

2 Model capa de persistencia

2.1 Diagrama



2.2 Descripció

2.2.1 Controlador Persistència

Com el seu nom indica, és el controlador de la capa de persistència. És l'encarregat de la comunicació amb la capa de domini per gestionar la càrrega i descàrrega de fitxers de disc. És a dir, gestiona la carrega i descarrega d'informació de la sessió com puguin ser els índexs o alguns atributs d'expressions booleanes o de l'espai vectorial i també gestiona els documents que es pengen i que es consulten.

La funció de llegir retorna un ArrayList amb títol, autor i contingut. El que fa és veure el format del fitxer que s'ha de llegir a partir del filepath que rep i en funció d'això crea un ParseTXT, ParseKITTY o ParseXML per retornar aquesta informació.

La funció de guardar document a memòria s'utilitza per mantenir els documents entre sessions a memòria principal. Sempre guardarà els documents en format txt i el que fa és crear un ParserTXT i guardar a la carpeta ./documents el fitxer amb el titol, autor i contingut que li passem com a paràmetres.

La funció de llegir s'utilitza tant per quan volem veure el contingut d'un dels documents que tenim guardats a l'aplicació (dins la carpeta ./documents) o per carregar un document que es trobi en qualsevol altra directori de l'ordinador. Si el document és d'un dels tres formats suportat es llegirà i s'afegirà al conjunt de documents de l'aplicació.

També tenim una funció per descarregar documents a alguna carpeta de l'ordinador. Se li passa el format i en funció d'aquest format crea un tipus de parser o un altre que serà l'encarregat de guardar a disc. La diferència entre aquesta funció de guardar i l'esmentada anteriorment és que aquesta es fa servir quan l'usuari vol descarregar-se un document i l'altra només es fa servir internament per anar guardant els documents que es van creant, modificant o eliminant mentres el programa està obert. Podríem haver fet servir aquesta darrera funció per realitzar les dues tasques però van considerar que era més clar tenir una funció específica per guardar a memòria els documents que estan carregats a l'aplicació perquè en aquest cas no necessitàvem ni filepath ni format (sempre es ./documents i format txt) i així evitàvem possibles errors i era més senzill

per fer canvis com podria ser canviar la carpeta on es van guardar tots els documents, ja que només faria falta canviar el path en aquesta funció.

També tenim la funció d'eliminar document, que únicament es fa servir per eliminar documents de la carpeta on emmagatzemem tots els documents que estan carregats a l'aplicació. Quan l'usuari decideix eliminar un document o modificar-lo es crida aquesta funció.

També tenim una funció loadObject i saveObject que permeten guardar qualsevol tipus d'objecte serialitzable. Aquestes funcions reben l'objecte en qüestió i el path on volem guardar-ho. Totes les estructures que guardem es trobaran dins del mateix directori: ./infoSessio. El que es guardarà serà informació de: idxObres, idxAutors, espai vectorial i consultes booleanes. Com per exemple el nom de les consultes booleanes guardades, o l'índex amb els autors i les obres que hi ha al sistema en el moment de tancar la sessió, etc.

Per poder obrir i descarregar aquesta informació no podem guardar directament una instància de la classe, ja que perdríem completament l'encapsulament. El que es fa és guardar estructures genèriques de Java com puguin ser ArrayList o TreeMaps de manera que les classes que han de guardar o recuperar la informació a disc seran les encarregades de codificar o descodificar aquesta informació per tal de recuperar la informació de la sessió anterior. Les úniques classes que hem permès que es guardin són les de la carpeta utils. Més concretament, hem fet servir Pair, Pes i TrieNode i els hem fet serialitzables.

2.2.2 Parse

És la classe abstracta que s'utilitzarà per parssejar els documents que es vulguin carregar o descarregar. Aquesta classe deriva en tres classes aplicant polimorfisme per poder llegir i guardar en els 3 formats que es permeten al programa: .kitty, .txt, .xml.

Tots tres *parsers* reimplementen les funcions de llegir document i guardar document. Són classes que no tenen atributs, únicament realitzen accions (llegir i escriure).

Les funcions del Parse són:

- **llegeixDocument(String filepath) : ArrayList**
    - Donat un filepath que inclou el nom del fitxer que es vol llegir i la seva extensió (.txt, etc) es retorna un ArrayList de tres posicions amb el títol, l'autor i el contingut. Si el fitxer que es vol llegir no és de cap dels tres formats suportats es retorna un ArrayList buit.
- **guardaDocument(String filepath, String titol, String autor, String contingut) : void**
    - En aquest cas el filepath és el nom de la carpeta on volem guardar el document. Títol, autor i contingut és la informació del document que guardarem. Com a decisió de disseny havíem decidit que els documents es guardessin com a *titol_autor.format*. Possiblement canviem la funció de controlador persistència que crida a guardarDocument per fer que en el filepath també s'inclogui el nom que volem que tingui l'arxiu. D'aquesta forma podem permetre que l'usuari guardi el document amb el nom que vulgui.

Al llegir ens assegurem que hi hagi títol i autor. En cas contrari, saltarà una excepció. Si al parssejar el document si no troba el contingut (per exemple no existeix l'etiqueta) farà com si fos contingut buit "".

2.2.3 ParseKITTY

És el *parser* del format propietari creat per a aquest projecte. Aquest format es diferencia en que emmagatzema la informació per blocs, s'indica l'inici d'un i fins que no s'indiqui l'inici del següent bloc tota la informació pertanyerà a aquest bloc. Es diferencia dels altres formats en que l'ordre en que es guardin les diferents parts del document és indiferent, ja que queda totalment encapsulada per cada secció que s'hagi definit. Per indicar les seccions s'utilitza *#nom de la secció#*. Per exemple: #titol#.

L'últim bloc del fitxer .kitty no estarà delimitat per cap secció posterior (és l'última). El contingut d'aquest darrer bloc serà tota la informació des de l'etiqueta fins al final del document.

Un exemple de document en format .kitty seria:

```
#titol#
Això és una mostra
#autor#
grup31.2


#contingut#
Aquí aniria el contingut. Al ser la darrera secció tot el text des d'aquí fins al final pertanyerà a l'últim bloc. Els salts de línia que hi ha després de l'autor també s'interpreten com si fos part del bloc, ja que el bloc s'acaba quan ja no hi ha més text o quan comença la següent etiqueta.
```

La funció de llegir documents permet llegir les tres etiquetes en qualsevol ordre. No hi ha una limitació del sistema tal i com passa amb el .txt. Podem començar escrivint el bloc de contingut o el de títol o el d'autor indiferentment i al parssejar es retornarà l'ArrayList amb la informació a la posició correcte: títol, autor i contingut.

2.2.4 ParseTXT

Es el *parser* del format .txt. Per llegir un document s'assumeix que a la primera línia es guarda l'autor, a la següent el títol i la resta de línies formaran el contingut. No hi ha cap forma d'identificar què és títol, autor o contingut excepte per la posició en la que es trobin en el fitxer (primera línia autor, etc).

Un exemple de document en format .txt seria:

```
Això és una mostra
grup31.2
Aquí aniria el contingut.
Evidentment, pot ocupar varies línies
```

L'enunciat del projecte ja ens limita la forma en què s'emmagatzema la informació al fitxer txt. La primera línia serà per l'autor, la següent pel títol i tota la resta pel contingut.

2.2.5 ParseXML

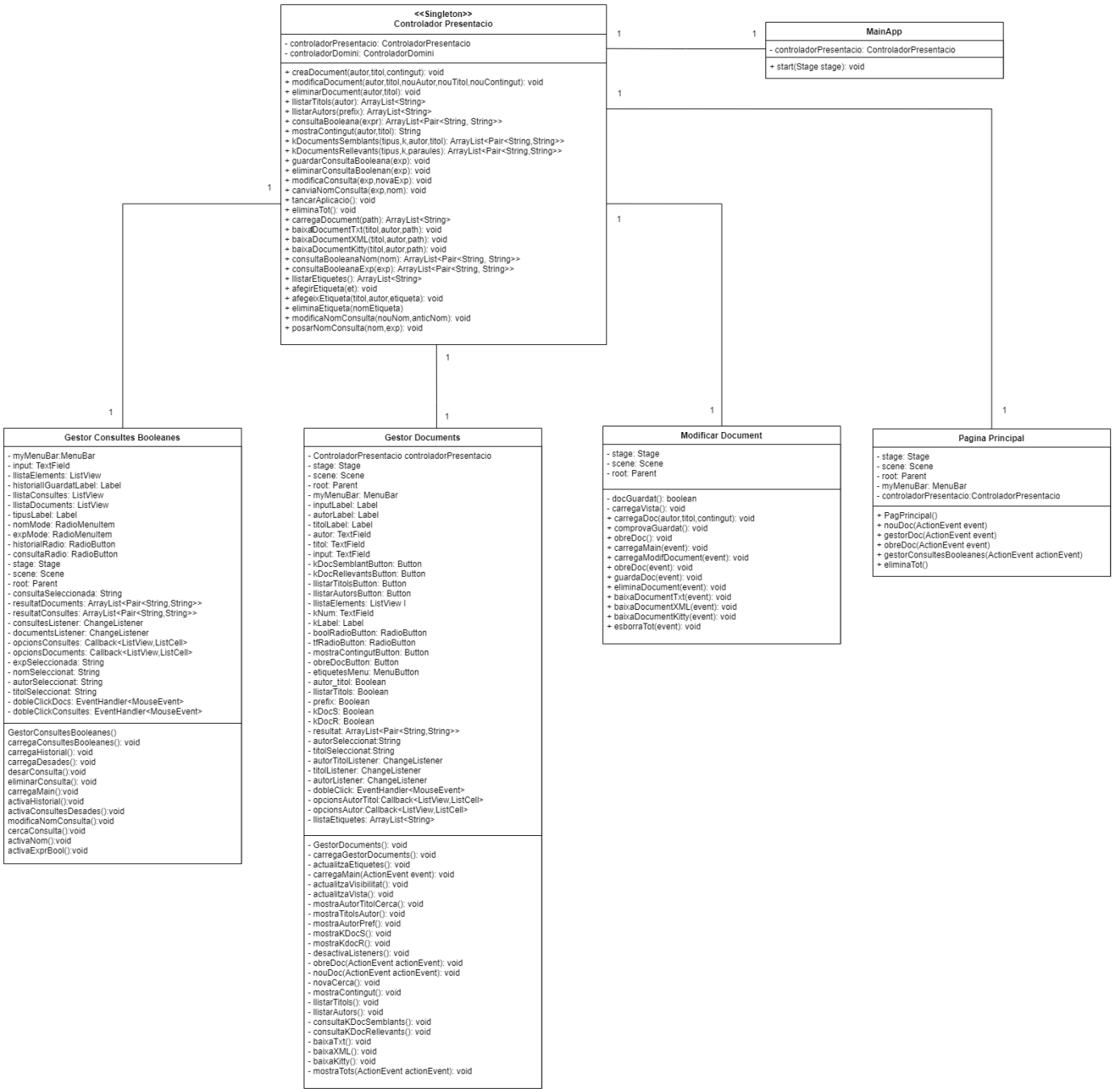
Es el *parser* del format .xml. Com a mínim els fitxers han de contenir l'etiqueta <titol> (sense l'accent), <autor> i <contingut>. Poden haver-hi altres etiquetes però seràn ignorades pel *parser*, excepte en el cas en què es trobin dins d'una de les tres etiquetes mencionades, que passarien a formar part del text de l'etiqueta. És a dir, si entre les etiquetes <titol> i </titol> hi ha una altra etiqueta, per exemple <extra>, l'etiqueta extra es considerarà que és part del títol i formarà part del string.

Un exemple de document en format .xml seria:

```
<titol>Això és una mostra</titol>
<autor>grup31.2</autor>
<contingut>Tot això formaria part del contingut. Si jo
ara poso una etiqueta <extra> hola </extra> apareixerà
parssejat com a part del contingut. Quan es mostri el
contingut es veurà una part que serà: "<extra> hola </extra>"
Fins aquí arriba el contingut</contingut>
```

3 Model capa de presentació

3.1 Diagrama



3.2 Descripció

3.2.1 Controlador Presentació

Es la classe encarregada de la comunicació entre les accions que realitza l'usuari i la capa de domini, on es realitzen els càlculs, les modificacions i la resolució de consultes.

Ara, comentarem les funcionalitats que l'usuari pot activar a cada vista, que a través del controlador, es comunicaran amb la capa de domini.

3.2.2 Pàgina principal

Presenta els grans grups d'opcions que pot voler escollir l'usuari. Serveix per encaminar-lo cap a la vista adequada tant si vol fer una consulta com si vol gestionar documents. A més presenta alguna acció molt habitual com crear un nou document per agilitzar l'ús de l'aplicació evitant navegar innecessariament pels menus. Com a comentari general per totes les vistes, sempre hi ha un botó per tornar a la finestra anterior per facilitar la navegabilitat. Com a detall adicional em afegit la possibilitat d'anar al GitLab del projecte a través d'un element interactiu a l'aplicació.

La pàgina principal ens presenta una interfície amb els següents botons que ens condueixen a les diferents vistes:

- **Nou Document:** Ens dirigeix a la vista de ModificarDocument de manera directa. Al tractar-se d'una acció molt habitual, evitem que l'usuari hagi de passar per GestorDocuments.
- **Carrega Document:** Com l'acció anterior, es tracta d'una operació molt habitual, de manera que actua de *shortcut*. L'usuari escull un document i aquest es carrega al sistema i se li pregunta a l'usuari si el vol modificar, en cas afirmatiu, es carrega la vista de ModificarDocument amb les dades que hi ha al document seleccionat. En cas que el document ja existís, se li pregunta a l'usuari si el vol sobreescriure.
- **Gestor Documents:** Aquest botó adreça a l'usuari la vista dedicada a la gestió de documents.

- **Consultes Booleanes:** Si es prem aquesta opció, se li presenta a l'usuari la vista per la gestió de consultes booleanes.

3.2.3 Gestor Documents

Es tracta de la vista encarregada de presentar les opcions disponibles per a la gestió de fitxers, a més d'operacions de consulta sobre els mateixos. Per defecte, la vista mostra els camps d'autor i títol per consultar el contingut del document. Aquesta part de la vista, canviarà en funció de la consulta que es vulgui fer mitjançant un menú desplegable o bé des de la llista de resultats fent click dret i escollint el tipus de cerca.

Ara, detallarem cadascuna de les opcions que presenta aquesta vista a través de botons per a la gestió de documents:

- **Etiquetes:** Aquesta funcionalitat del sistema permet a l'usuari organitzar els fitxers per temes o categories i si accedeix a través d'aquesta interfície. Al presionar aquest botó es presenten les opcions d'afegir i eliminar etiquetes en forma de menú desplegable. A més, quan les etiquetes queden registrades es mostren en aquest menú desplegable perquè l'usuari pugui seleccionar les que ell vol que estiguin actives.
- **Mostra Tots:** En premer aquest botó es mostren tots els documents registrats al sistema en una llista interactiva detallant l'autor i el títol de cadascun. En aquesta interfície si fem clic dret sobre un document podem escollir entre mostrar el seu contingut, fer una consulta de k documents semblants a aquest, eliminar-lo o assignar-li alguna de les etiquetes prèviament definides. Si fem doble clic esquerre se'ns dirigirà a la vista ModificarDocument amb els camps de títol, autor i contingut emplenats. Finalment, comentar que al entrar a la vista de GestorDocuments es realitza aquesta opció, és a dir, es mostren tots els documents registrats a la llista interactiva.
- **Obre Document:** Si seleccionem un document de la llista i cliquem aquest botó, se'ns dirigirà a la vista de ModificarDocument amb els camps de títol, autor i contingut emplenats, com fa la funcionalitat del doble clic sobre el document.
- **Nou Document:** Condueix a la vista de ModificarDocument.

- **Baixa:** Ens permet baixar en algun dels tres formats suportats (TXT,XML,KITTY) el document que haguem seleccionat de la llista interactiva. El document es guarda on hagi indicat l'usuari.

Comentem ara les opcions de consulta que presenta el desplegable "Filtre":

- **Autor-Títol:** Com hem comentat abans, és la consulta per defecte que ens permet veure el contingut d'un document donat el seu títol i l'autor.
- **Llistar títols autor:** Mostra els titols dels documents escrits per un autor a la llista interactiva.
- **Llistar autors per prefix:** Com el seu nom indica, mostra els autors el nom del qual comença pel prefix consultat. A més, en mostrar-se el noms dels autors que satisfan la consulta, amb clic dret sobre el mateix podem consultar els títols dels seus documents que apareixen com element interactiu per poder fer les consultes abans descrites, mantenint fluïdesa en la gestió de documents per una millor usabilitat.

  Aquestes consultes de llistar poden ordenar-se alfabèticament o per la data d'última modificació del fitxer.

- **K documents semblants:** Presenta una interfície amb camps autor títol per indicar el document pel que es vol fer la consulta. Com passa amb les altres consultes, aquests camps s'autocompletarien si indiquem el document seleccionat-lo a la llista prèviament. Una vegada definit el document podem indicar el nombre de documents que volem que ens retorni la consulta, a més de poder indicar l'estratègia de pesos que es vol fer servir per a la consulta. El resultat de la consulta s'entrega a la llista interactiva.
- **K documents paraules:** Presenta una funcionalitat similar a l'anterior, però en aquest cas l'element de consulta no es un document, sino que es una llista de paraules. De la mateixa manera que abans podem indicar quants documents volem que ens retorni la consulta així com l'estratègia de pesos utilitzada.

3.2.4 Modificar Documents

Aquesta vista s'especialitza en la creació, carrega i eliminació de documents del sistema. Passem a comentar les seves funcionalitats principals començant pels desplegables i les seves opcions:

- **Fitxer:**
    - **Nou document:** És la interfície per defecte a l'accedir a la vista ModificarDocuments. Permet introduir les dades als camps d'autor, títol i contingut. Al prémer el botó "Desar", aquest document quedarà enregistrat al sistema. A més, a través del botó "Baixa", podem descarregar-nos el fitxer que acabem de crear al qualsevol dels formats disponibles. Finalment, si s'intenta desar un document que ja existia avisa al usuari de si el vol sobreescriure i si es vol crear un nou document sense guardar sobre el que s'està treballant també se li notifica a l'usuari perquè decideixi si el vol guardar o no.
    - **Carrega Document:** Aquesta opció obre una finestra del gestor de fitxers del sistema operatiu per tal de seleccionar fitxers existents a disc que volen afegir-se a l'aplicació. En cas que ja existeixi un document identificat de la mateixa manera a l'aplicació, s'avisarà a l'usuri de si vol sobreescriure el fitxer. Una vegada carregat els seus camps es mostren a la vista per si es volen modificar.
    - **Guardar:** Compleix la mateixa funcionalitat que el botó "Desa"
    - **Baixa Document:** Compleix la mateix funcionalitat que el botó "Baixa" explicat a l'opció "Nou document".
    - **Elimina Document:** Permet a l'usuari eliminar el document que es trobi als camps d'autor i títol que presenta la vista. Com a element de seguretat es demana confirmació a l'usuari abans d'eliminar el document.
    - **Menú Principal:** Opció que es troba a les diferents vistes, al mateix desplegable, per permetre a l'usuari tornar a la pàgina principal millorant la navegabilitat per les vistes.

- **Edita:** Ens dona l'opció d'esborrar tots els camps del document que estigui obert en aquell moment.

3.2.5 Gestor Consultes Booleanes

Aquesta vista és l'encarregada de tot el que te a veure amb consultes booleanes, tant fer-les com gestionar-les. Queda separada de la vista GestorDocuments, en permetre aquesta gestió d'expressions booleanes que la distingeixen com una funcionalitat amb més entitat propia.

Comentem les seves funcionalitats principals començant pels desplegables i les seves opcions:

- **Tipus d'input:** Ens permet escollir entre introduir directament l'expressió booleana (opció per defecte) o si en canvi volem introduir el nom d'una expressió ja guardada al sistema.

Quan es realitza una consulta el resultat es mostra a una llista interactiva com la de les altres vistes on si el fa doble clic a algun dels documents es canviarà la vista a GestorDocuments. El resultat de la consulta es pot ordenar alfabèticament pel títol o per l'autor.

En segon lloc, hi ha una segona llista interactiva que mostra l'historial de consultes que s'han realitzat. Si l'usuari vol, pot fer clic dret sobre la consulta per decidir si la vol guardar al sistema, esborrar-la o modificar-la o doble clic perquè s'escrigui al camp d'expressió booleana per poder fer la consulta. Finalment aquesta segona llista interactiva es pot canviar per la llista de consultes desades. Aquestes es mantindran al finalitzar la sessió de la mateixa manera que les guardades al historial, pero amb un nom identificatiu.

3.3 Implementació

Per crear les vistes hem utilitzat JavaFX. Cada vista està formada per un fitxer FXML, amb l'esquelet bàsic. Cadascun està associat a una classe java on estan tots els mètodes que es necessitem per fer el programa funcional.

3.3.1 Elements útils

- Listener: Serveix per estar atents a la interacció de l'usuari amb els components gràfics. Un exemple d'ús, està a GestorDocumets, on cada cop que se selecciona un document de la llista, s'omplen els camps d'autor i títol els corresponents valors del document.
- Alert: Són diàlegs molt simples que ens permeten mostrar diferents tipus d'informació (Informació, confirmació, warning...) en un popup.
- FileChooser i DirectoryChooser: Permeten seleccionar el path, ja sigui d'un document que volem carregar al sistema, com el directori on volem guardar un document.
- Callback: És cridat cada cop que fem click dret a un element de la llista, mostra les opcions que es poden fer amb ell.
- Altres: Elements més bàsics, com etiquetes, per mostrar text; botons, per activar certs mètodes o modes; camps per introduir text; entre d'altres.

4 Estructures de dades i algorismes

4.1 Estructures de Dades

4.1.1 Espai Vectorial

L'espai vectorial ha estat la forma d'emmagatzemar els documents per la seva comparació, que ha proposat l'enunciat. Aquesta estructura ens permet guardar els documents com a una cadena de les seves paraules amb un pes associat. D'aquesta seqüència de paraules s'eliminen les *stopwords* i també es poden eliminar els nombres, les paraules que només apareixen en un document així com els signes de puntuació.

4.1.2 Trie

Un trie és un tipus d'arbre molt utilitzat per trobar cadenes de paraules.

Cada node d'un trie té assignat un caràcter i conté un punter per cada lletra que segueix la del node actual i un indicador de si la lletra actual és fi de paraula o no. El node arrel no correspon a cap lletra, els seus fills seran les primeres lletres de les paraules.

A l'hora de buscar les paraules es va mirant lletra per lletra, una paraula no existirà si per dues lletres consecutives a i b, b no és fill d'a, o si no està el final de paraula marcat.

Aquesta estructura de dades ens és molt útil a l'hora de buscar els autors que comencen per algun prefix.

4.1.3 Map

TreeMap

Els autors es guarden en un TreeMap. D'aquesta forma podem llistar als autors alfabèticament. La key de cada element del map serà el nom de l'autor i el value serà un altre TreeMap amb totes les obres d'aquell autor. En aquest segon TreeMap la key serà el títol de l'obra i el value serà el DocumentPROP amb la informació de l'obra d'aquell autor. Més endavant ha

LinkedHashMap

Aquesta implementació de map ens permet mantenir l'ordre d'addició facilitant el manteniment de l'espai vectorial pel que fa a la interrelació entre *Weights*, TAS i *Norms*. També s'utilitza al passar consultes de similaritat pel mateix motiu, mantenir la coherència i evitar posibles desparellaments de dades.

SortedMap

Aquest tipus de map ens permet ordenar les claus de forma creixent. Ens és útil la l'hora d'emmagatzemar els nodes del Trie, de manera que quan demanem els autors, ens els dóna ordenats.

4.1.4 List

Una llista és una estructura molt útil si el que volem és iterar sobre un conjunt d'elements. Permet tenir elements repetits.

ArrayList

És una llista a la que se li pot modificar la mida, per tant, és molt útil per llistes a les que li volem afegir o eliminar elements, com a les expressions booleanes o a les frases d'un document.

4.1.4 Pair

Es tracta d'una classe auxiliar que ens ha permès simplificar algunes estructures de dades. Al no existir com a tal a *Java*, hem decidit implementar-la nosaltres amb un ús identic al seu homònim a C++.

4.2 Algorismes

4.2.1 Per a consultes booleanes

Tractament de les negacions

Per tractar les operacions NOT, cada expressió booleana es guarda un atribut que ens diu si està negada o no. Durant la funció treureNot(), si troba una NOT que afecta a tota l'expressió la treu i inverteix l'atribut negat. Això ens permet poder seguir operant amb l'expressió sense la NOT i saber si estava negada o no.

Algorisme per resoldre les consultes booleanes

L'estratègia que seguim per resoldre cada consulta es basa en Dividir i Vèncer. Dividim cada consulta en consultes més petites fins a arribar a un cas trivial i després anem ajuntant els resultats fins a obtenir de nou la consulta inicial.

Per començar intentem treure els parèntesis i les NOT que afectin a tota la consulta.

Comprovem si la consulta és bàsica, és a dir, conté només una paraula, un conjunt de paraules entre claudàtors o una seqüència de paraules entre cometes. Si ho és la resolem i retornem.

A continuació intentem separar-la en dos per una OR exterior, i si no n'hi ha cap, per una AND. Sempre ens trobarem en algun d'aquests dos casos, ja que si no, o la consulta era incorrecte o la hem resolt just abans.

Després cridem a resoldre les dues subconsultes de manera recursiva.

Finalment, si l'operador era una OR retornem la unió de les dues subconsultes, si era una AND retornem la intersecció. En el cas que l'expressió original estigués negada s'intercanvien les AND per OR i les OR per AND. Al separar la consulta de dos en dos ens assegurem que el resultat seguirà sent correcte tot i estar negada (Per les lleis de De Morgan).

4.2.2 Per a l'espai vectorial

Similaritat del cosinus

Aquesta representació dels documents com a vectors de pesos ens serveix per poder fer consultes de similaritat amb altres documents o amb conjunts de paraules. Per fer-ho utilitzem l'algorisme de similaritat del cosinus que surt de la idea de donar l'angle que separa dos documents, l'angle que ens representa com de diferents són dos conjunts de paraules.

$$\cos(\theta) = \frac{d \cdot q}{||d|| \cdot ||q||}$$

Aquesta és la fórmula amb la que treballem, s'utilitza el cosinus en comptes de l'angle, ja que és més fàcil de calcular. La d representa el vector de pesos del document, la p el vector de pesos de la *query* que pot ser un altre document i no té perquè utilitzar la mateix estratègia d'assignació de pesos. El producte del numerador es defineix d'aquesta manera:

$$d \cdot q = \sum_{i=1}^{N} w_{i,d} \cdot w_{i,q}$$

És a dir, el sumatori dels productes entre el pes de cada paraula al document i a la *query*. En el denominador tenim el producte de les normes que es calculen d'aquesta manera:

$$|q| = \sqrt{\sum_{i=1}^{N} w_{i,q}}$$

És a dir l'arrel quadrada de la suma dels quadrats dels pesos de la *query* o del document depenent de la norma que vulguem calcular.

En el nostre cas, computem la similaritat del cosinus per donar els k documents més similars a un donat o per donar els k documents més rellevants donat un conjunt de paraules. Aquestes consultes es poden fer amb l'estratègia d'assignació de pesos booleana o la de tf-idf pels documents, escollint a la hora de fer la consulta. El conjunt de paraules que son les consultes, se'ls hi assignen pesos amb l'estrategia booleana.

Tf-idf

Finalment parlem d'aquesta estratègia d'assignació de pesos que té en compte si una paraula és freqüent però també si ho és a través de tots els documents. D'aquesta manera paraules molt freqüents com "cosa" no obtindran tanta rellevància com si ho farien si només tinguéssim en compte la freqüència local a cada documents. Aquesta estratègia busca potenciar la importància de les paraules no tan freqüents que moltes vegades són les que ens permeten diferenciar el contingut de dos textos.

La fórmula per calcular els pesos és aquesta:

$$tf\_idf = tf(t,d) \cdot idf(t,D)$$

- $tf(t,d) = f_{t,d}$, es pot definir com el nombre de vegades que apareix una paraula al document , i ha diverses maneres de quantificar el *tf* però aquesta és la més àgil de calcular junt amb la booleana (1 si està 0 si no).
- $idf(t,D) = log\frac{N}{1+|\{d \in D : t \in d\}|}$, es defineix com el logaritme del quocient entre el nombre total de documents i el nombre de documents on apareix *t*. L'1 del denominador està per si el mot no apareix a cap document.

5 Canvis segona entrega

5.1 Consultes Booleanes

Hem reescrit completament el codi de Consultes Booleanes per intentar paral·lelitzar-lo.

Hem eliminat les classes ConjuntConsultesBooleanes i ExpressioBooleana i hem afegit les següents: NouConjuntExpressionsBooleanes, NouExpressioBooleana, Node i ControladorConsultesBooleanes.

5.1.1 Totes les classes Consultes booleanes

Ara generem explícitament l'arbre d'una expressió booleana, i hem volgut canviar pràcticament l'estructura de totes les classes per fer el codi més entenedor i poder implementar "fàcilment" la paral·lelització. Tot i això, la majoria de mètodes, estructures de dades i algorismes els hem pogut reutilitzar. Hem afegit també la possibilitat d'afegir un nom a les expressions booleanes, cosa que ens ha fet canviar l'estructura concreta on guardàvem expressions (però mantenint el tipus. p ex: de LinkedList hem passat a LinkedHashMap, que és el mateix però amb clau-valor).

5.1.2 Paral·lelització

El perquè de la paral·lelització: Estem creant un gestor de documents, per tant és normal pensar que durant l'ús hi haurà molts documents guardats. Però per una restricció del plantejament ens demanen que els documents estiguin sempre guardats a disc, per si n'hi ha molts, volem canviar el disc per una base de dades en una futura extensió... Per tant totes les operacions que necessitin recorrer tots els documents hauran d'accedir un cop a disc per documents. Com sabem, les operacions de lectura a disc son "molt lentes", i si n'hi ha moltes seguides haurem de fer esperar a l'usuari. La solució que hem trobat és fer-les en paral·lel.

Hem considerat 3 maneres diferents de paralelitzar el codi.

La primera i més bàsica es tracta de crear un sol thread que executi tots els càlculs en paral·lel mentre l'usuari pot seguir interactuant amb el thread principal. Quan s'afegeix/modifica/elimina un document es crea un thread que recalcula els resultats de totes les consultes booleanes guardades. Si tornen a haver-hi canvis en els documents, el thread principal s'esperarà i quan el secundari acabi tornarà actualitzar de nou les consultes alliberant el principal.

La segona idea és que diferents threads resolguin una sola consulta booleana. Al tenir-ho guardat en forma d'arbre podem fer que cada node creï dos threads per resoldre les subexpressions i quan acabin ajunti els resultats. Una possible millora directa és fer que en comptes de cada node crear dos threads, ell mateix resolgui una part i només creï un thread per l'altra. D'aquesta manera dividim entre dos el nombre de threads que es creen. Tot i així, aquesta idea pot explotar molt fàcilment, si la consulta és una mica llarga. I tampoc guanyem gaire, ja que cada thread fulla s'haurà de recórrer tots els documents. De fet és molt pitjor que la primera idea.

Per la tercera decidim, en comptes d'utilitzar un thread per node de l'arbre, tenint el problema de crear-ne masses, utilitzar un nombre predefinit (T) de threads i que cadascun comprovi (N/T) documents, essent N el nombre total de documents. D'aquesta manera aconseguim controlar el nombre de threads que creem, dividir les operacions realment costoses i nombroses (lectura de fitxers a disc) i no malgastar memòria i temps tenint threads que es passen gairebé tota la estona esperant a altres. A més aquesta idea també aconsegueix el mateix que la primera, ja que el thread principal l'única feina que farà serà crear (T) threads, i per tant l'usuari podrà seguir fent coses mentre es calculen les consultes guardades. Si hi ha nous canvis mentre s'estan executant els threads, s'aturen tots i es creen de nou amb els canvis posats.

Ens hem decantat per la tercera opció, ja que és la més completa sense ser molt difícil de programar.

Quin nombre de threads utilitzem?

La majoria d'ordinadors moderns de gamma mitjana (Intel i5 de 10ª gen o més i AMD Ryzen 5 o més) tenen 12 o més threads. Nosaltres hem escollit utilitzar un màxim de **10 threads** per resoldre les consultes booleanes. Hem volgut maximitzar aquest nombre suposant que el programa s'executarà en un ordinador com els descrits al principi. No té sentit posar-ne més, ja que l'ordinador no serà capaç d'executar-los paral·lelament, i per tant a partir d'aquest nombre màxim no millorarà el rendiment. De fet empitjorarem el rendiment si augmentem massa el nombre de threads ja que s'hauran de fer moltes més unions de resultats.

5.1.3 Possibles millores:

En comptes d'accedir a cada document un cop per expressió booleana, accedir-hi un cop i resoldre totes les expressions booleanes que s'hagin de resoldre. Això implicaria canviar l'algorisme de cerca en documents i l'estructura dels threads que hi ha actualment.

Escollir automàticament el nombre màxim de threads en funció de les especificacions de l'ordinador on s'executa l'aplicació.

5.2 Espai Vectorial

5.2.1 Estructura

Hem modificat la manera com emmagatzemaven els pesos de les paraules de cada document registrat al sistema. Ara, en comptes d'estar limitat a dos tipus de pesos diferents per paraula per la manera com s'havia implementat, es poden afegir els tipus de pesos que es vulgui en futures actualitzacions.

Per fer-ho s'ha creat una nova classe "Pes" abstracta amb un mètode polimòrfic "calcPes" de la que es pot heretar per implementar cada tipus de pes diferent, en el nostre cas "PesBool" i "PesTfIdf". Les noves classes que s'afegeixin només han d'implementar la seva manera de calcular els pesos al mètode "calcPes" i afegir aquest nou pes a l'espai vectorial.

Finalment, ara les paraules es tracten de manera *caseinsensitive* perquè l'assignació de pesos sigui més coherent.

5.2.2 Possibles millores

El càlcul dels pesos d'un nou document o d'una modificació d'un document, pot arribar a ser costos si tenim documents amb un vocabulari molt extens o un gran nombre de documents. A més, en cas que s'afegissin noves estratègies pel càlcul de pesos que fossin més complexes aquest problema no faria més que empitjorar. Per això seria interessant utilitzar l'estratègia de la paral·lelització mitjançant *threads*.

En el cas de l'espai vectorial utilitzaríem la primera estrategia descrita a consultes booleanes, és a dir, un sol *thread* que executi tots els càlculs en paral·lel mentre l'usuari pot seguir interactuant amb el *thread* principal, de manera que l'actualització de pesos interrumpeixi el mínim possible l'ús de l'aplicació.

5.3 IdxObres

En aquesta versió índex obres ja no guarda els documents. Aquesta funció la realitza exclusivament la capa de persistència. Tot i això hem mantingut la classe perquè ens és útil per realitzar consultes i per conèixer l'estat de l'aplicació i els documents al que hauria de tenir accés.

5.3.1 Estructura

Al no guardar el document l'estructura que hem fet servir és una mica diferent. En comptes de ser un Map<String, Map<String, DocumentPROP>> ha passat a ser un TreeMap<String, Set<String>>. El key del map és el nom de l'autor i el value és un set amb totes les seves obres. Fem servir un set per poder consultar més ràpidament si un autor té una obra. Amb aquesta estructura a part d'accedir ràpidament als autors, com és un Tree també ens permet retornar els autors ordenats alfabèticament. D'aquesta forma quan vulguem visualitzar la llista de tots els documents apareixeran ordenats alfabèticament pel nom de l'autor. Dins de les obres d'un mateix autor també apareixeran llistades alfabèticament. La forma en què s'ordena alfabèticament és primer les lletres majúscules i després les minúscules.

Com a decisió de disseny vam acordar que es permetien autors amb majúscules i amb minúscules i es considerarien autors diferents. L'alternativa hagués estat transformar els títol i l'autor a *lower case* però vam decidir respectar el mateix mètode que implementen els sistemes operatius de manera que dos documents amb el mateix nom però un en majúscules i l'altre en minúscules es consideren fitxers diferents. Per tant, joan i Joan per al nostre sistema seran autors diferents. I al ordenar alfabèticament primer apareixerà Joan i després joan (l'ordre alfabètic és A, B, ..., Z, a, b, ..., z).

5.3.2 Possibles millores

Estem pensant en afegir una millora de cara a l'entrega final. La idea és afegir la data de consulta o creació o modificació dels documents (el darrer cop que s'ha interactuat amb

ells) de manera que puguem ordenar per data. Creiem que aquest canvi pot facilitar la feina als usuaris, ja que entenem que els documents amb els que més interactua són aquells que vol tenir més a mà. Si ordenem els documents pel criteri de la data apareixerien els més recents abans.

Per implementar-ho hem pensat en canviar el set de strings que conté les obres per un map de <string, data>. D'aquesta forma es segueix complint la condició de que no hi hagi dues obres amb el mateix nom i a la vegada ens podem guardar la data.

Únicament caldria fer petites modificacions en les funcions de la classe (inserir en un map o en un set no és igual) i caldria afegir una funció que ordenés totes les obres i autors per data. A part, s'haurien de canviar les funcions que retornen els títols i autors del sistema per a que també retornessin la data. Per exemple, canviant el Pair per un ArrayList de tres posicions.

5.4 Persistència

Hem afegit la capa de persistència. Això ha fet que idxObres ja no emmagatzemi el document amb la informació. Aquesta tasca passa a ser responsabilitat de persistència. Cada cop que es vulgui consultar la informació relativa a un document primer es comprovarà que el document existeixi i després es demanarà aquesta informació a persistència, la qual aplicarà polimorfisme en funció del format del document per retornar el títol, autor i contingut. Hem fet que persistència només retorni aquests tres paràmetres per mantenir millor l'encapsulament i l'estructura en tres capes, ja que una opció hagués estat emmagatzemar i retornar instàncies de Docu. Però Docu és una classe de Domini i volíem mantenir les classes separades. Com l'única informació imprescindible és la de títol, autor i contingut; la resta de càlculs ja els podem fer a Domini tal i com fèiem a la primera entrega. Per tant, només estem involucrant classes generals de Java i no les classes que s'utilitzaran a domini. A més, així facilitem que en un futur es pugui modificar o afegir noves funcionalitats. Per exemple si volem guardar un resum del document únicament hauríem de passar a domini un quart String amb aquesta informació.

5.5 Etiqueta

Hem pensat en afegir com a funcionalitat extra les etiquetes. La idea original era tenir l'opció de marcar els documents com a Favorits però vam decidir estendre aquesta idea més enllà i permetre la possibilitat de tenir diferents etiquetes que ens permetin agrupar els documents per tipus o temàtiques segons li convingui a l'usuari.

5.5.1 Estructura

Per fer-ho hem creat la classe Etiqueta i a Conjunt Documents hem afegit una estructura que ja hem explicat anteriorment per emmagatzemar totes les etiquetes i els documents que tenen cadascuna d'aquestes etiquetes.

5.5.2 Decisions de disseny

Se'ns va acudir més d'una manera d'implementar aquesta funcionalitat. Una opció hagués estat afegir a idxObres a la part dels títols de cada autor posar un ArrayList amb les etiquetes a les que pertany aquell document. Hagués pogut estar una bona solució en el cas que les etiquetes fossin sempre les mateixes i no permetéssim afegir noves etiquetes ni eliminar-ne ni canviar el nom. Com nosaltres volíem permetre totes aquestes funcionalitats vam descartar la idea, ja que es feia molt difícil canviar el nom d'una etiqueta o eliminar-la, ja que hauries de recórrer totes les obres de l'índex i mirar quines etiquetes tenia per modificar-les o per llistar-les. També es feia molt complicar filtrar els documents per etiqueta de forma que només es mostressin per pantalla els documents d'una certa etiqueta, ja que calia recórrer novament totes les obres.

Hem preferit guanyar en senzillesa i eficiència a costa de tenir emmagatzemada més informació de l'estrictament necessària. De la forma en què ho hem fet si un document té alguna etiqueta apareixerà en l'estructura que emmagatzema els documents que contenen l'etiqueta (un idxObres). De manera que si té dues etiquetes apareixerà en dues instàncies d'Etiqueta a més d'aparèixer en el conjunt de totes les obres que guardem a conjunt documents. Però d'aquesta forma és molt senzill afegir noves etiquetes i canviar els noms o eliminar-ne. També és més senzill llistar les obres que tenen una etiqueta o llistar totes les etiquetes.

A més, assumim que si tens molts documents la majoria tindran com a molt una única etiqueta, si es que en tenen. Per tant, només estaríem duplicant la informació (cada document apareixeria a l'idxObres del conjunt de documents i tornaria a aparèixer a l'índex de l'etiqueta que se li hagi assignat).