Documentació del projecte

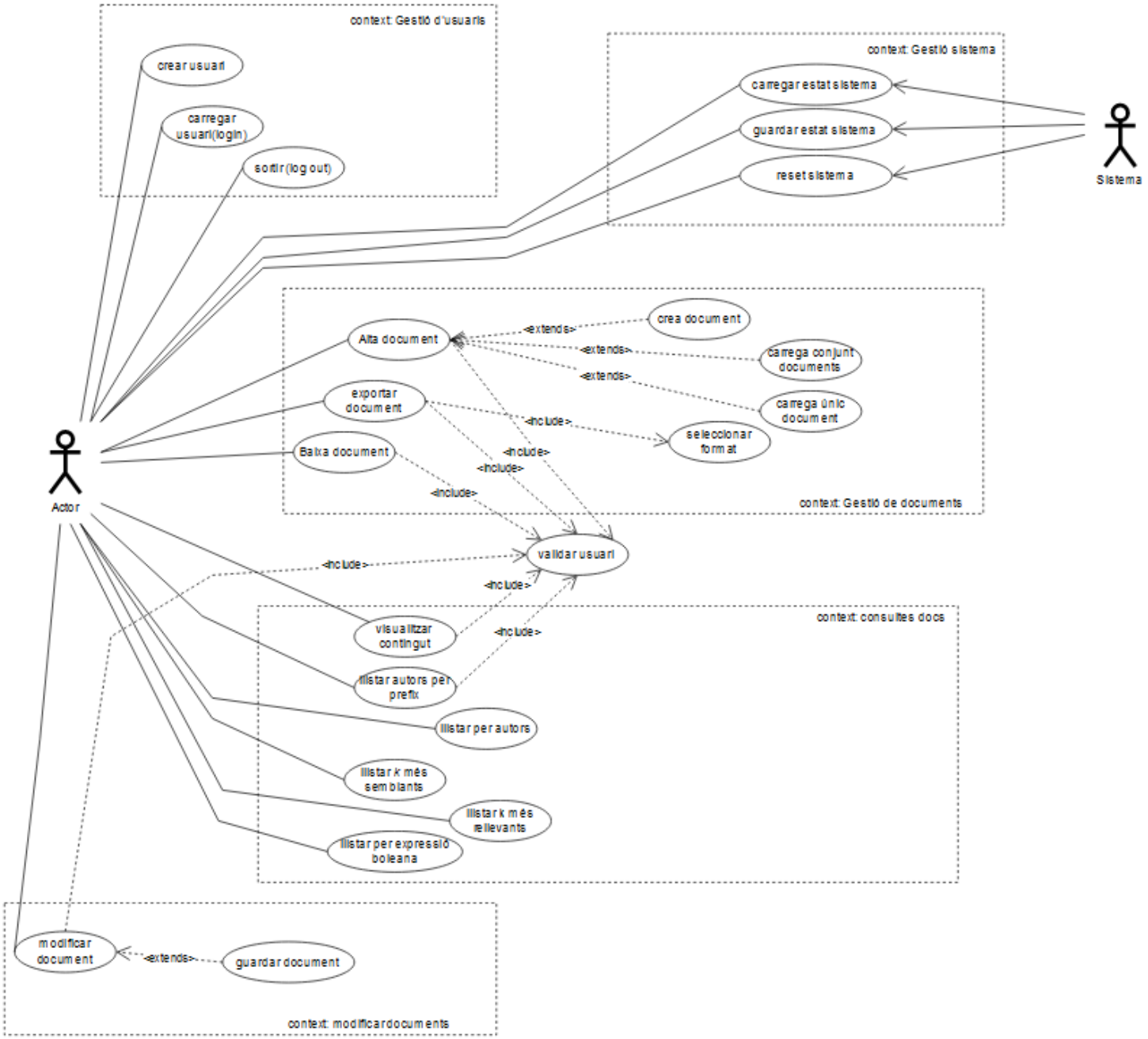
Albert Vidal González
Ignasi Juez Guirao
Lluís Llull Riera
Iván Serrano Hernández

GRAU-PROP Q1 2022/23

SUMARI

1. Casos d'ús

1.1 Diagrama casos d'ús



1.2 Descripció casos d'ús

context: Gestió usuari

Crear Usuari

Descripció: Un actor extern al sistema crea un usuari amb un cert nom i paraula de pas.

Pre: el nom de l'usuari no està repetit

Post: s'enregistra un nou usuari identificat per un cert nom, pel que guarda una paraula de pas. L'usuari enregistrat no està carregat (no figura que hagi fet *log in*)

Carregar Usuari

Descripció: Es carrega en el sistema un usuari

Pre: l'usuari ha de figurar en el sistema, és a dir, prèviament s'ha d'haver creat. La paraula de pas i el nom de l'usuari han de ser correctes.

Post: L'usuari figura en el sistema com que ha fet *log in*.

Sortir Usuari

Descripció: Operació per sortir del perfil actual.

Pre: l'usuari ha de figurar en el sistema i a més ha d'estar prèviament carregat.

Post: L'usuari deixa de figurar en el sistema com que ha fet *log in*, però continua figurant en ell.

Validar Usuari

Descripció: El sistema valida si l'usuari existeix i està prèviament carregat, perquè després aquest pugui realitzar altres interaccions.

Pre: L'usuari existeix i està prèviament carregat.

Post: Es passa a realitzar la interacció que ha disparat al *validar usuari*.

context: gestió de documents

Exportar document

Descripció: Operació per guardar o exportar un document del nostre sistema intern a la memòria externa, en un format seleccionat (.txt o .xml)

Pre: El document existeix i l'usuari està validat

Post: S'exporta a la memòria externa del programa el document amb el format seleccionat.

Seleccionar format

Descripció: Operació per seleccionar un format per un document (.txt o .xml).

Pre: null

Post: Retorna el format seleccionat entre els següents: txt o xml.

Baixa document

Descripció: Operació per eliminar un document existent del nostre sistema intern.

Pre: El document existeix i l'usuari està validat.

Post: Es suprimeix tota la informació del document corresponent.

Alta document

Descripció: S'afegeix un nou document al sistema (pot ser creant-lo o carregant-lo).

Pre: L'usuari ha d'estar validat. No existeix cap document amb el mateix títol i autor.

Crear document

Descripció: Es crea el document des de zero.

Pre: L'usuari ha d'estar validat. No existeix cap document amb el mateix títol i autor.

Post: S'afegeix al sistema el nou document creat

Carrega únic document

Descripció: Carrega d'un sistema extern un únic document

Pre: L'usuari ha d'estar validat. No existeix cap document amb el mateix títol i autor.

Post: S'afegeix al sistema de l'usuari el document.

Carrega conjunt documents

Descripció: Carrega d'un sistema extern un conjunt de documents.

Pre: L'usuari ha d'estar validat. No existeix cap document amb el mateix títol i autor.

Post: S'afegeix al sistema de l'usuari tots els documents del conjunt.

context: Modificar documents

Modificar document

Descripció: Operació per a modificar el contingut d'un document. L'usuari determina si vol guardar el document modificat amb l'acció Guardar document modificat.

Pre: El document existeix i l'usuari està validat.

Post: S'actualitza el nostre sistema de dades amb el document modificat. En cas que l'usuari guardi el document, es dispara l'operació *Guardar document modificat.*

Guardar document modificat

Descripció: Operació per guardar un document un cop ha estat modificat. Operació disparada per Modificar document

Pre: El document existeix i ha estat modificat anteriorment

Post: Es guarden els canvis de totes les modificacions realitzades al document en el sistema intern, també es modifica la data de darrera modificació. En cas de cancel·lar-se l'operació no es guarden els canvis.

S'actualitzen les estructures de dades per les consultes de documents.

context: Consultes en documents

Visualitza contingut

Descripció: Ens mostra el contingut d'un document donat un títol i l'autor.

Pre: El document identificat pel títol i autor existeix i l'usuari ha estat validat.

Post: Retorna el contingut del document corresponent.

Llistar autors per prefix

Descripció: Ens mostra una llista d'autors comencen amb un prefix concret.

Pre: L'usuari ha estat validat.

Post: Retorna la llista d'autors que comencen amb un prefix concret.

Llistar per autor

Descripció: Operació que llista els títols donat el nom d'un autor.

Pre: El nom de l'autor ha de ser vàlid.

Post: Es llisten tots els documents que pertanyen a un cert autor.

Llistar k més semblants

Descripció: Ens mostra els k documents més semblants a un document D

Pre: El document D, identificat per un títol i autor existeix.

Post: Retorna el títol i l'autor dels k documents més semblants a un document D

Llistar per expressió booleana

Descripció: Donades una expressió booleana formada pels operadors & | i ! i conjunts de paraules (delimitats per {}), seqüències de paraules (delimitades per “”), o paraules soltes com a operands, s'obtenen tots els documents que contenen una frase que satisfà l'expressió booleana.

Pre: L'operació booleana és vàlida

Post: Es retorna el conjunt de documents que compleix l'expressió booleana

Llistar k més rellevants

Descripció: Donat un conjunt de paraules p, ens mostra els *k* documents més rellevants en base *p*

Pre: Nul.

Post: Retorna el contingut dels *k* documents més rellevants en funció de *p*.

SECCIÓ DE COMENTARIS

comentari per la primera entrega

Per aquesta primera entrega on únicament s'exigeixen les funcionalitats principals hem decidit no implementar res de les funcionalitats relacionades amb la gestió d'usuaris, per tant cap dels casos d'ús d'aquest context s'han portat a terme al codi del projecte (cosa que si es farà per les pròximes entregues).

A més, pel mateix motiu, tampoc s'ha implementat res relacionat amb la funcionalitat d'exportar un fitxer, ja que per aquesta entrega no s'exigeix res relacionat amb la capa de persistència.

2. Model Conceptual

2.1 Diagrama model conceptual

Consulta els nostres diagrames al directori /UMLS.

2.2 Descripció de les classes del model conceptual

La descripció detallada de cadascun de les classes i els seus mètodes i atributs està feta al javadoc, en aquest apartat únicament explicarem el propòsit de cada classe, per tal de donar una visió general de com hem estructurat el projecte. D'aquesta manera el codi no queda obstruït per tota la documentació.

2.2.1 Capa de Domini

Classe Usuari

Representa l'estructura d'un Usuari, aquest és identificat pel seu Username. És una superclasse abstracta que pot ser implementada per la seva subclasse UsuariLoggedIn.

Classe Usuari Logged-In

Representa un Usuari que ha iniciat sessió amb les seves credencials. La seva superclasse és Usuari i té els mateixos atributs.

Classe Conjunt Documents

Representa al conjunt de documents del sistema, es podria dir que és la classe principal. Conté tant els documents com la representació interna del conjunt (en forma d'un objecte de la classe Inverted File), i concentra la majoria de funcionalitats principals.

Classe Document

Representa l'estructura d'un Document, que s'identifica pel seu Títol i Autor. La classe Document guarda el contingut *raw* d'aquest en un objecte de tipus contingut, i a més guarda una breu representació interna d'aquest, formada únicament pels types dels documents (paraules sense repeticions, sense caràcters especials i sense stopwords).

El propòsit d'aquesta representació interna és el de poder facilitar l'actualització de les estructures de dades del *Inverted File* i el *Vector Term Model* cada cop que s'elimina o modifica un document.

Classe Contingut

La classe Contingut conté el contingut d'un document. Per no guardar el document sencer com a un únic String (ja que podria ser problemàtic per qüestions de gestió de memòria) , es guarda com una llista de Strings on cadascun és un paràgraf.

Classe Inverted File

Aquesta classe conté la representació interna del conjunt de documents. En ella es troben les ED's[1] del Fitxer Invertit, en forma d'un HashMap on les claus són els noms dels documents i els valors objectes de la classe wordData, i del vector Space Model, en forma d'un altre HashMap on les claus són els noms dels documents i els valors objectes de la classe fileTermData.

Les ED's d'aquesta classe són les responsables de gestionar de forma eficient les cerques, per tant, s'actualitzen a cada operació que modifica el conjunt de documents (afegir un de nou, modificar-ne o esborrar).

Classe FileTermData

Aquesta classe representa el que seria el *term vector* d'un document. Guarda tots els types d'aquest, i per cadascun el seu pes.

Pel pes de cada paraula, es té un objecte de tipus weight, on es separa el pes de la paraula en el terme *idf* i *tf*.

Classe wordData

Aquesta classe representa el que seria la *posting list* d'una paraula al inverted File. Conté un HashMap on la clau és una paraula i el valor la seva llista de posicions al document, a més de les funcions necessàries per obtenir dades d'aquestes paraules, com per exemple el seu nombre de repeticions, afegir i esborrar paraules,etc.

Classe Operació booleana

Representa una expressió booleana formada pels operadors & | i ! i conjunts de paraules.

Classe Controlador de Domini

Representa el controlador del domini, que concentra funcionalitats principals del projecte i les comunica amb la capa del domini.

[1] ED: Estructura de Dades

2.2.2 Capa de Presentació

Classe altaDoc

Classe que proporciona als usuaris donar d'alta un document o un conjunt de documents mitjançant les funcionalitats implementades. Disposa per a carregar documents en conjunt o individuals, crear un txt o fins i tot un xml.

Classe carregaDoc

Classe que dona la possibilitat de carregar documents en conjunt o individuals des del seu ordinador als usuaris mitjançant una interfície interactiva.

Classe cerca

Classe que ofereix un ventall de possibilitats dels tipus de cerca que hem implementat perquè l'usuari disposi de les cerques més acurades possibles amb les seves característiques úniques per cada una d'elles.

Classe cercaAutors

Classe que proporciona a l'usuari un tipus de cerca on el resultat d'aquesta seran tots aquells autors que comencin pel prefix introduït per l'usuari.

Classe cercaBooleana

Classe que permet a l'usuari realitzar una cerca booleana mitjançant la seva expressió booleana. Aquesta haurà d'estar construïda correctament, formada pels operadors & | i ! (amb les normes de precedència habituals i la possibilitat de patentitzar per canviar aquesta precedència) i conjunts de paraules (delimitats per {}), seqüències de paraules (delimitades per ""), o paraules soltes com a operands. Altrament, no es realitzarà la cerca i el resultat serà un missatge informatiu. Disposa d'una gran varietat d'algoritmes per a l'ordenació de totes les cerques anteriors que hagi dut a terme l'usuari.

Classe cercaKRellevants

Classe ofereix a l'usuari una cerca per obtenir els k documents més rellevants (en quant a contingut) per a la query introduïda per l'usuari. Aquesta query ha d'estar formada per p paraules (denotades col·lectivament com a query), i un enter k.

Classe cercaSimilars

Classe que proporciona a l'usuari una cerca per obtenir els k documents més semblants a un document D existent en la base de dades de l'aplicació mitjançant el seu títol i autor a elecció de l'usuari.

Classe cercaTitAut

Classe que dona la possibilitat a l'usuari de cercar el contingut d'un document existent en la base de dades de l'aplicació mitjançant el seu títol i autor.

Classe cercaTitUnAut

Classe que permet a l'usuari consultar tots els títols d'un autor mitjançant el nom de l'autor. Si aquest no existeix el sistema informarà a l'usuari.

Classe confirmacionEsborratTot

Classe que informa l'usuari que ha esborrat tot el conjunt de documents de l'aplicació. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe confirmacionEsborrat

Classe que informa l'usuari que ha esborrat el document seleccionat. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe confirmacionSalida

Classe que informa l'usuari de si està segur que vol tancar l'aplicació per evitar errades humanes. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe confirmacioSortidaToLogin

Classe que informa l'usuari de si està segur que vol tancar la seva sessió per evitar errades humanes. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe confirmaTornarPunt

Classe que informa l'usuari de si està segur que vol restaurar el punt de restauració seleccionat per evitar errades humanes. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe creaDoc

Classe que permet a l'usuari crear un nou document mitjançant un títol, un autor i el seu respectiu contingut. Aquest no serà creat si no es proporciona un títol i un autor.

Classe creaDocXml

Classe que permet a l'usuari crear un document en format XML. Aquest no serà creat si no es respecten les especificacions indicades per a la creació d'un document XML.

Classe EliminaDoc

Classe que informa l'usuari de si està segur que vol eliminar el document seleccionat per evitar errades humanes. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe elminarUsuaris

Classe que permet a l'administrador de l'aplicació eliminar un Usuari introduint-hi el nom de l'usuari que desitja eliminar.

Classe error

Classe que ens permet implementar tots els errors que es poden donar per errades humanes i d'aquesta manera mantenir a l'usuari informat d'aquest.

Classe exportaDoc

Classe que proporciona una interfície gràfica per a poder exportar el document seleccionat per l'usuari. Aquest ha de seleccionar el nom del fitxer que desitja exportar i el directori del seu ordinador a on vol exportar-lo.

Classe exportarTots

Classe que proporciona una interfície gràfica per a poder exportar tot el conjunt de documents de l'aplicació. Aquest ha d'indicar el directori del seu ordinador a on desitja exportar el conjunt.

Classe gestorRestauracions

Classe que proporciona una interfície gràfica a la qual només l'administrador hi té accés per a poder gestionar els punts de restauració de l'aplicació.

Classe login

Classe que ens trobem només iniciem el programa, on l'usuari validarà les seves credencials i passarà a la classe *menú* o bé, si encara no té un compte, tindrà l'opció de registrar-se, que ho farà accedint al registre d'usuaris on donarà un nom d'usuari, que haurà de ser únic i una contrasenya que compleixi els requisits de seguretat.

Classe menu

Classe principal un cop s'ha fet login, per a usuaris normals com per administrador. Aquesta classe en serveix de punt d'entrada a totes les funcionalitats implementades que pot realitzar un usuari com donar d'alta documents, modificar documents o fer cerques.

Classe modificacioDocument

Classe que llista tots els documents gestionats per l'aplicació. Permet a l'usuari modificar o esborrar els documents que té permisos per fer-ho mitjançant el seu títol i autor.

Classe modificarDoc

Classe que ens permet modificar el contingut d'un document determinat. Aquesta classe és només accessible un cop el document ha sigut creat i no abans.

Classe modificarDocCreador

Classe que permet a l'usuari modificar el contingut del document seleccionat pel mateix si aquest té els permisos adequats.

Classe mostraConfirmMod

Classe que informa l'usuari que el document que vol modificar ha estat modificat correctament. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe mostraNomesAdmin

Classe que informa l'usuari que només l'administrador de l'aplicació hi té accés. D'aquesta manera mantenim a l'usuari informat sobre les limitacions que té en l'aplicació.

Classe mostraJaExisteix

Classe que informa l'usuari que el document ja existeix en l'aplicació. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe mostraOKCreat

Classe que informa l'usuari que el document s'ha creat correctament. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe mosrtaOKexportat

Classe que informa l'usuari que el document seleccionat s'ha exportat correctament. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe noExisteixAutor

Classe que informa l'usuari que l'autor introduït no existeix en l'aplicació. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe noExisteixDoc

Classe que informa l'usuari que el document introduït no existeix en l'aplicació. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació

Classe nouPuntRestauracio

Classe que permet a l'administrador crear un nou punt de restauració de l'aplicació mitjançant un nom únic respecte els altres punts de restauració prèviament creats.

Classe registrarUsuari

Classe que proporciona la interfície necessària a l'usuari perquè creï el seu usuari mitjançant un nom que ha de ser únic i una contrasenya.

Classe resultatCercaAutors

Classe que mostra a l'usuari el resultat de la cerca d'autors donat un prefix proporcionat pel mateix. Disposa d'un parell d'algorismes per a l'ordenació del resultat de la seva cerca.

Classe resultatCercaBooleana

Classe que mostra a l'usuari el resultat de la cerca booleana donada una expressió booleana pel mateix. Disposa d'un parell d'algorismes per a l'ordenació del resultat de la seva cerca.

Classe resultatCercaContingut

Classe que mostra a l'usuari el resultat de la cerca del contingut del document seleccionat pel mateix. Aquest es podrà llegir però no modificar.

Classe resultatCercaRellevants

Classe que mostra a l'usuari el resultat de la cerca dels k documents més rellevants (pel que fa a contingut) per a la query introduïda per l'usuari.

Per la implementació de la funció, els documents que resulten de la cerca s'ordenen alfabèticament pel seu nom i autor, i no per la seva similitud.

Classe resultatCercaSimilars

Classe que mostra a l'usuari el resultat de la cerca dels k documents més semblants a un document D existent en la base de dades de l'aplicació mitjançant el seu títol i autor a elecció de l'usuari.

Per la implementació de la funció, els documents que resulten de la cerca s'ordenen alfabèticament pel seu nom i autor, i no per la seva similitud.

Classe resultatCercaTitolsUnAutor

Classe que mostra a l'usuari el resultat de la cerca de tots els títols d'un autor donat el seu nom. Disposa d'un parell d'algorismes per a l'ordenació del resultat de la seva cerca.

Classe tornarPuntRestauracio

Classe que permet a l'administrador tornar a un punt de restauració de l'aplicació creat anteriorment.

Classe totsOKexportats

Classe que informa l'usuari que el conjunt de documents de tot el sistema s'han exportat correctament. D'aquesta manera mantenim a l'usuari informat sobre els canvis que es realitzen en l'aplicació.

Classe controladorPresencia

Classe controlador que ens relaciona totes les classes de la capa de presentació entre elles i amb la resta de classes mitjançant els altres controladors implementats. Està connectada amb el controlador de domini per intercanviar informació amb les classes d'aquest. Per implementar tots aquestes classes d'aquesta capa hem utilitzat la llibreria, swing i AWT.

2.2.3 Capa de Persistencia

Per aquesta capa s'ha de mencionar que no s'ha seguit el model tradicional d'arquitectura en 3 capes, sino que s'han diluït les operacions que serien pròpies del controlador de persistència en el controlador de la capa de domini.

En un projecte que fes servir bases de dades, els mètodes de la capa de persistencia serien nombrosos i segurament els més crítics de la plataforma, però aquest no és el nostre cas, ja que l'emmagatzemament persistent de les dades l'hem realitzar d'una manera molt més simple.

Com s'ha emmagatzemat l'estat de la plataforma d'una sessió a una altra

Per dur a terme aquesta tasca, inicialment vam pensar en escriure en fitxers de text pla el contingut de cadascun dels objectes que voliem preservar d'una sessió a una altra, que serien els següents:

- Contingut *raw* dels documents
- Representació interna dels documents
    - Fitxer invertit
    - *file term vector*

El principal problema amb aquesta via era que era necessari recórrer cadascuna de les estructures anteriors, escrivint el contingut en fitxers de text pla de forma que després es pogués llegir, i això era molt poc pràctic, especialment pel tema de l'escalabilitat (qualsevol canvi en l'estructura de les classes dels elements que s'emmagatzemen comportaria alterar la forma d'escriure i llegir contingut dels fitxers).

Per aquest motiu, vam decidir fer la persistència del nostra projecte a base de serialització. Gràcies a l'estructura del projecte, on totes les classes giren entorn a la de *ConjuntDocuments.java* ens podriem aprofitar de les característiques de la serialització i guardar l'estat de tota la plataforma serialitzant únicament un objecte, el del *ConjuntDocuments.java.*

Per dur a terme aquesta tasca, desde el controlador de domini en el moment de sortir de la plataforma es serialitza aquest objecte en un fitxer (*metadata.ser*), que es deserialitza cada cop que es fa *login*.

Avantatges de la serialització

Els principals avantatges de serialitzar objectes en comptes de escriure el seu contingut en fitxers és el següent:

- Escalabilitat: Gràcies a la serialització, realitzar canvis en l'estructura de classes no afecta a la forma de llegir i escriure, pel que el procés de guardar l'estat de la plataforma és pràcticament independent a la seva implementació (excepte per un matís explicat més endavant).
- Senzillesa: El procés de llegir i escriure un objecte en un fitxer en forma de *bytestream* és infinitament més simple que el d'anar escrivint/llegint el seu contingut.
- Compatibilitat: Un fitxer amb un objecte serialitzat pot ser des-serialitzat en qualsevol màquina sense cap mena de problemes.
- Seguretat: Aquesta és possiblement la característica més important de totes. El fet de codificar el contingut de la plataforma en un *bytestream* fa que els documents que hi han en aquesta únicament puguin ser accessibles des de dins de la pròpia plataforma.

Desavantatges de la serialització

El principal inconvenient que ens hem trobat de cara a implementar la persistència de la forma descrita és la impossibilitat de fer servir camps estàtics com a atributs de les classes que voliem preservar d'una sessió a una altra.

Degut a que la serialització només pot fer servir l'estat de l'objecte, degut a que els camps estàtics no pertanyen a l'objecte sino a la classe, aquests no poden ser conservats entre sessions.

Per aquest motiu, ens ha resultat impossible implementar un controlador de Persistència, ja que hi havia forma possible que aquest i el controlador de domini poguessin accedir al mateix objecte de *ConjuntDocuments.java* sense que aquest tingués camps estàtics, per tant hem inclós les operacions de serialització i des-serialització al controlador de domini (i no al controlador de persistència, que és on haurien de ser).

3. Estructures de dades i algorismes

En aquest projecte, el desenvolupament de totes les classes han girat entorn a les estructures de dades necessàries per gestionar les cerques de forma eficient; el fitxer invertit i el model d'espai vectorial.

A continuació es fa una breu descripció de les estructures de dades i algorismes utilitzats i la seva justificació.

3.1 Estructures de Dades

3.1.1 El conjunt de documents

El conjunt de documents es podria dir que és la classe principal del projecte, ja que és on es concentra la gestió de les funcionalitats principals.

Un objecte de la classe *conjuntDocuments* conté dues estructures de dades fonamentals:

- El conjunt de Documents del sistema, representat a través d'un HashMap on la clau d'aquest és el nom del document i el *value* és un objecte de la classe Document.
- La representació interna del conjunt de documents del sistema, que és un objecte de la classe Inverted File.

El motiu per al que hem triat el HashMap (per l'ED ja esmentada i per gairebé tota la resta del projecte) és que les operacions més comunes i que més es realitzen, com per exemple *get()*, *put()* o *containsKey()*, s'executen en temps constant per a una funció de hash decent com pot ser la que fa servir Java per defecte, on el codi d'una certa key es computa de la següent forma per un String.

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

On *n* és la longitud del *string* i *s[i]* el i-èssim caràcter d'aquest.

A diferència de qualsevol altra estructura de dades, com per exemple seria un TreeMap, on aquestes mateixes operacions es realitzen el *O(log n)*.

Per tant, el conjunt de documents conté dues coses principalment; els documents tal com són, sense tractar el seu contingut, d'altra banda, té la representació interna d'aquest, gràcies a la qual es poden gestionar les operacions de cerca de manera eficient.

3.1.2 El fitxer invertit

El fitxer invertit és una estructura que guarda per cadascuna de les paraules del sistema de fitxers les següents dades (el que es coneix com a *posting list)*:

- Document on apareix la paraula
- Per cada document:
    - La llista de posicions on apareix

Components del fitxer invertit

Per una qüestió d'eficiència en les cerques, tal com ja s'ha mencionat, hem decidit fer servir un HashMap per emmagatzemar cada paraula (clau de l'arbre) i la seva *posting list* (valor).

Per facilitar les operacions amb les posting list d'una certa paraula, hem decidit crear una nova classe, a la que hem anomenat *wordData*, on s'emmagatzema per cada paraula una parella del tipus *<String, List<integer>>* que representa el document i la llista de posicions on la paraula apareix pel document donat.

En aquesta classe hi han definides una sèrie d'operacions bàsiques com per exemple són afegir/esborrar una nova entrada per la *posting list* d'una paraula, o obtenir la llista de documents on apareix, entre d'altres.

Justificació de l'ús d'aquesta ED

Per exemple, en el cas de comprovar si existeixen un conjunt de paraules en tots els documents, utilitzant l'*Inverted File*, el cost és únicament el cost de cerca del hashMap, és a dir constant O(1), ja que podem accedir directament el conjunt de documents on apareix una paraula, en canvi, si no tinguessin aquesta estructura de dades hauríem de recórrer tots els documents i a l'hora recórrer totes les paraules de cada document per veure si apareix o no, és a dir el cost seria quadràtic en la mida del conjunt de documents.

3.1.3 El model d'espai vectorial

Components de la ED

El model d'espai vectorial és una eina que ens permet calcular el grau de similaritat entre documents a partir de l'assignació de pesos a les paraules que cadascun conté.

Per a aquesta assignació de pesos, s'ha seguit el model de tf-idf (*term frequency - inverse term frequency* per les seves sigles), on el pes d'una certa paraula es calcula fent servir l'algorisme *tf-idf*.

Per gestionar això, hem decidit tenir a la classe Inverted File (la representació interna del conjunt de documents), una ED del tipus *HashMap* on la *key* és el nom del fitxer, i el valor és un objecte de la classe *fileTermData*, que conté els pesos de cadascuna de les paraules que formen un document.

L'objecte de la classe *filetermData* conté una ED del tipus HashMap, on la *key* és una paraula i el *value* és un objecte que conté el pes de la paraula, a més d'algunes operacions bàsiques sobre aquest; retornar el pes, recalcular-lo, actualitzar el pes, etc.

El motiu principal pel qual hem decidit tenir el pes d'un cert terme emmagatzemat en un objecte d'una classe, i no com un simple *float*, és perquè a l'hora d'actualitzar el pes, si es descompon com el producte de *idf*tf*, únicament és necessari recalcular la component *idf*, donat que la part *tf* només es recalcula en el cas concret en que s'hagi modificat el contingut del document.

Justificació de l'ús de la ED

L' ús d'aquesta estructura de dades està pensat per realitzar de forma eficient totes les operacions en les que sigui necessari calcular la similitud entre documents, per a que en el moment de servir una *query* no sigui necessari realitzar cap càlcul més que el de la similaritat, tenint ja tota la ED construïda prèviament.

Un cas en el que aquesta ED proporciona grans avantatges en quant al rendiment és per la cerca dels K documents més semblants, la nova funcionalitat implementada per la segona i tercera entrega.

L'ús del model d'espai vectorial redueix considerablement el cost d'aquesta operació, ja que al ja tenir calculats els *file term vectors* de tots els documents, en el moment de servir una query únicament és necessari construir el *term vector* dels termes de la query i realitzar el càlcul de la similaritat (cost constant) amb tots els documents (cost lineal), i per últim ordenar-los (cost logarítmic).

3.1.4 Document i el Contingut del document

Un document en el nostre projecte és un objecte que conté les següents dades:

- El seu contingut "raw", sense alterar: per facilitar les operacions de modificar i de veure el contingut
- Una representació interna reduïda: per quan és necessari actualitzar les ED's del fitxer invertit o dels *file terms*, per poder conèixer quines són les paraules que requereixen actualització (d'aquesta forma s'eviten càlculs innecessaris).

El contingut d'un document s'emmagatzema en una llista de *Strings*, on cadascun dels Strings representa un sol paràgraf. El motiu pel qual hem triat aquesta representació i no una altra, com podria ser guardar tot el contingut en un sol *String* directament per una qüestió de bona gestió de memòria. Per un document molt gran, guardar un String únic pel contingut podria exigir massa memòria adjacent, en canvi, amb el format triat només es necessita memòria contigua per cadascun dels paràgrafs.

3.2 Algorismes

Els algorismes més rellevants del projecte són els següents.

3.2.1 Algorisme tf-idf

Ha estat l'algorisme que hem fet servir per determinar el pes de les paraules en funció de dos factors:

- La seva freqüència en un cert document
- La seva freqüència en el conjunt de documents

Donat un String amb el contingut d'un document, l'algorisme emprat funciona de la següent manera.

1. Es preprocessa el String amb el contingut
    a. S'eliminen caràcters especials
    b. S'eliminen stopwords
2. Es troba quina és la paraula amb freqüència màxima dins del document (aquest càlcul s'aprofita per realitzar-lo durant l'addició dels types del document al fitxer invertit).
3. Per cadascun dels *types* al document, es troba quina és la seva freqüència al document i quina és la seva freqüència al conjunt del document aprofitant l'estructura de dades del fitxer invertit.

4. Un cop es tenen les dades anteriors, es pot calcular el pes de la paraula dins del document de la següent forma:

$$\mathrm{tf}(t,d) = \frac{\mathrm{f}(t,d)}{\max\{\mathrm{f}(t,d) : t \in d\}}$$

$$\mathrm{idf}(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \times \mathrm{idf}(t,D)$$

5. Amb el pes calculat, s'afegeix a l'estructura de dades corresponent del vector space model.

Per la forma de calcular el pes d'una paraula, concretament el terme de la *inverse document frequency*, es requereix que quan s'afegeixen, modifiquen o eliminen documents al sistema es recalculi la component *idf* de cadascuna de les paraules del sistema.

Aquest és un càlcul costós, ja que és lineal en el nombre de paraules en el sistema, per tant, no és realista pensar que en un sistema a gran escala es pugui fer per cada moviment. Nosaltres hem decidit que pel nostre sistema sí que sigui així, doncs el nombre de documents carregats és molt baix per aquesta primera entrega i no fer aquest càlcul a cada moviment podria traduir-se en diferències significatives en els resultats.

3.2.2 Algorisme de la cerca booleana

Per a resoldre la cerca booleana hem emprat recursivitat a causa de les prioritats que estableixen els parèntesis ens les consultes dels usuaris. I tenint en compte les comprovacions de la veracitat de les expressions booleanes escrites pels mateixos.

Per a les comprovacions mirem si l'expressió booleana està ben construïda en termes de parèntesis i brackets. Per aconseguir això primer trèiem tot menys els parèntesis i brackets del String que conté l'expressió booleana d'aquesta manera podem comprovar que aquests estan ben construïts. Un cop tenim només els parèntesis i brackets hem implementat una solució basada en Stack on cada parèntesi i bracket tindrà la seva parella i s'anirà buidant el Stack. Si l'expressió està ben construïda aquest serà buit, altrament aquest serà no buit.

Un cop hem comprovat la veracitat de l'expressió ens toca avaluar l'expressió i donar un resultat. Per a fer-nos més fàcil aquesta avaluació hem decidit fer una funció que ens crei una estructura basada en Set a partir de l'expressió booleana, ja que fer les OR i AND en aquesta estructura de dades és molt simple. Aquesta funció l'hem anomenada exchangeVals i per cada petita consulta ho substitueix pels documents que la compleixen, la qual ens retorna un vector de sets.

Per exemple una cerca que volgués buscar els documents on no aparegui la paraula *despues* i si aparegui la seqüència de paraules *hola* es veuria així:

!{despues} & (“hola”) ens retornaria: [!][doc1][&][(][doc2 doc3][)]

On doc1 hi apareix la paraula *despues* i on doc2 i doc3 apareix la seqüència *hola*.

A partir d'aquí utilitzem aquesta estructura que ens ha retornat *exchangeVals* per a resoldre les expressions. Tenim 3 funcions per a resoldre-ho *S*, *P* i *A*.

- *S* és la funció principal, serà la que ens retorni el resultat final i realitza les operacions AND i OR.
- *P* és la funció que ens permet fer les negacions
- *A* és la funció que ens permet fer els parèntesis amb prioritats si n’hi ha en l’expressió, si no retorna un resultat.

Es fa una crida a la funció *S* i aquesta primerament fa una crida a *P* on aquesta funció si es tracta d’una negació (!) retorna el conjunt total de documents menys el resultat de la crida a *A*. Altrament retorna la crida a *A*. *A* comprova si es tracta d’un parèntesi, si ho és aquesta fa una crida a *S* repetint el cicle que acabem de descriure. Altrament retorna el resultat. El resultat arriba a *S* i aquesta realitza llavors les operacions AND i OR, fent una crida de nou a *P*.

Ho il·lustrem amb un exemple simple, ídem per expressions més complexes:

Definim el contingut de 4 documents per a l'exemple.

doc1: hola prueba

doc2: si despues

doc3: hola

doc4: si despues

!{despues} & (“hola”)

exchangeVals -> [!][doc2 doc4][&][(][doc1 doc3][)]

S rep l’expressió [!][doc2 doc4][&][(][doc2 doc3][)] i fa una crida a *P* on *P* mira si la posició 0 és o no una negació, en aquest cas si. Llavors s’agafa el conjunt total de documents i es fa una crida a *A*. Aquesta mira la posició 1, com no és un parèntesi retorna el valor de la posició, en aquest cas [doc2 doc4]. *P* rep aquest valor i els treu del conjunt total, ja que estem fent una negació, i ens queda [doc1 doc3]. Aquest valor és retornat a *S* on mira la posició 2 i és una AND, llavors fa una crida a *P* per fer la AND de [doc1 doc3] amb el que retorni *P*. *P* comprova la posició 3 i com no és una negació fa una crida a *A*. Aquesta comprova la posició 3 i al ser un parèntesi fa una crida a *S*. *S* fa una crida a *P*, *P* fa una crida a *A* i *A* comprova la posició 4 i al no ser un parèntesi retorna el resultat [doc1 doc3] a *P*, *P* a *S* i *S* a *A*. *A* comprova que la següent posició és un parèntesi, per tant, pot acabar i ho retorna a *P*, *P* a *S*. *S* fa la AND de [doc1 doc3] amb [doc1 doc3] i retorna el resultat que és: [doc1 doc3].

3.3 Noves funcionalitats respecte la primera entrega

3.3.1 Cerca dels k documents més rellevants

Per la segona i tercera entrega, hem decidit implementar la funcionalitat opcional de cercar els *k* documents més rellevants per una certa *query*.

Implementar aquesta funcionalitat no ha estat una tasca complicada, gràcies a la forma en que vam construir les principals ED's del projecte, el *file term vector* i el *Inverted File*, de forma que fossin fàcilment escalables en el cas d'afegir noves funcionalitats.

Per trobar els documents més similars a una certa *query* hem seguit un plantejament molt similar al de calcular la similitud entre documents, doncs el que hem fet ha estat tractar la query com si fos un document més, de forma que la majoria d'operacions les hem pogut reciclar de les funcionalitats ja implementades a la primera entrega.

(Per entendre millor el seu funcionament, consultar l'apartat 3.4.1)

3.3.2 Sistema d'usuaris

Tot i que per la primera entrega ja vam fer unes primeres pinzellades del que seria un sistema d'usuaris (únicament vam realitzar la funcionalitat de *login*), posteriorment hem decidit portar-ho més enllà.

La idea base del projecte era tenir un gestor de documents, que pogués realitzar les operacions bàsiques (altes, baixes i modificacions) i algunes més complexes (cerques, càlcul de similaritats, etc.) però únicament per un sol usuari. El que hem fet ha estat implementar dos tipus d'usuari; un de normal i un administrador, de forma que dins del *pool* de documents compartits, un usuari només pugui realitzar les operacions bàsiques ja mencionades sobre els seus documents, i les més complexes, les cerques, sobre tot el conjunt de documents.

D'aquesta manera, un usuari pot crear, borrar i modificar únicament els documents de la seva autoría, però pot realitzar cerques en qualsevol document, de forma similar a com seria en un sistema de fitxers compartit real.

Per altra banda, l'usuari administrador és qui pot fer canvis en el sistema que afectin altres usuaris. Per tant, l'administrador té les característiques implementades suficients per a poder eliminar el document de qualsevol usuari del sistema. L'administrador i només ell també disposa de la funcionalitat de poder eliminar tot el conjunt de documents que formen el sistema, d'aquesta manera netejar-lo i començar de nou. A més a més, també disposa de les funcionalitats suficients per a gestionar punts de restauració de l'aplicació que inclou crear punts de restauració i tornar a un cert punt de restauració creat anteriorment. Finalment hem afegit la funcionalitat de que l'administrador pugui eliminar usuaris del

sistema, els documents d'aquest no es borraran, pero l'usuari no podrà accedir amb les seves credencials.

Per aquestes funcionalitats de l'administrador hem creat en el menú un espai per a l'ús únic de l'administrador on només pot entrar ell, altrament l'usuari no administrador veurà un error informant-lo de que només l'administrador té permís.

A més a més, hem implementat una nova funcionalitat la qual permet que l'usuari pugui veure només els seus documents. D'aquesta manera millorem l'experiència de l'usuari en la nostra aplicació ja que, el conjunt de documents en el sistema pot arribar a tenir un volum ingent en determinades situacions.

Finalment per a facilitar la usança del nostre lliurament hem decidit implementar una nova funcionalitat per a que l'usuari pugui exportar tots els documents alhora, donant l'oportunitat a aquest per si desitja realitzar algun tipus de backup extern o exportació general.

3.3.3 Historial de modificacions del sistema

Per crear aquesta funcionalitat ens hem inspirat en el *Google Drive* (tot i que de forma més simple, òbviament), on per una carpeta compartida entre diversos usuaris, qualsevol d'ell pot consultar els canvis que s'han realitzat en tot moment. I això és precisament el que hem fet.

Cadascuna de les operacions bàsiques que es realitzen (altes, baixes i modificacions de documents) es registren en un fitxer de text pla, que qualsevol dels usuaris (independentment del seu estatus) poden consultar en qualsevol moment.

3.3.4 Algoritmes d'ordenació dels resultats de les cerques

Per aquesta segona entrega hem millorat l'experiència de l'usuari en les cerques, més concretament en *cerca per títols d'un autor, cerca autor* i la *cerca booleana.*

Per la primera cerca, al mostrar el resultat, ara l'usuari té l'oportunitat d'ordenar el resultat. Aquest resultat són els títol del autor que hagi introduït i aquest es poden ordenar alfabèticament ascendent o alfabèticament descendent.

Per a cerca autor tenim la mateixa implementació, l'usuari introdueix un prefix i ara tindrà l'oportunitat d'ordenar alfabèticament ascendent o alfabèticament descendent els autors que comencin per aquell prefix.

I finalment, per la cerca booleana ho hem elaborat més ja que ho permetia. En el historial de expressions booleanes utilitzades per l'usuari es poden ordenar de 6 formes diferents. Aquestes són les següents: ordre alfabètic, de més recent a menys, de més antic a menys, de més gran a menys, de més petit a menys i finalment normal que torna al resultat original. I no ens hem deixat la possibilitat d'ordenar els resultats també de la cerca booleana. Per aquest resultat hem trobat adequat tenir dues implementacions; ordre alfabètic per autors i ordre alfabètic per títols.

3.3.5 Punts de Restauració

Una funcionalitat que hem trobat molt interessant implementar ha estat la de crear punts de restauració. La idea és que l'administrador en qualsevol moment pugui decidir crear un punt de *backup* o restauració amb l'estat del sistema de documents en aquell moment, de forma que aquest quedi gravat en un fitxer de tipus *.ser* a l'igual que es fa amb la persistència.

D'aquesta manera, en qualsevol moment l'administrador pot decidir carregar un punt de restauració anterior (pel que prèviament haurà especificat el nom), i aquest es carregarà substituint l'estat del sistema en aquell moment.

Per implementar aquesta funcionalitat hem fet pràcticament el mateix que per conservar l'estat del sistema entre sessions, hem serialitzat l'objecte conjunt documents, només que en comptes de serialitzar-lo en un mateix fitxer (com és el cas de la persistència, amb el fitxer *metadata.ser*) es serialitza en un fitxer indicat per l'usuari.

NOTA: els fitxers on es graven els punts de restauració no poden estar repetits, i el nom d'aquests no pot tenir ni punts ni espais.

3.3.6 Creació de documents en formats alternatius

Per aquesta tercera entrega donem l'oportunitat a l'usuari de poder crear documents des de la nostra aplicació documents en els formats xml a més de text pla.

Per a cada format hi ha unes especificacions que s'han de complir per a la seva correcte creació, altrament no deixarà crear el document que desitgi l'usuari.

- Per a la creació d'un document del tipus txt, només serà possible la seva creació si aquest conté un títol i un autor, on el contingut és opcional.
- D'altra banda, per a la creació d'un document del tipus xml s'ha de realitzar de la forma definida dels documents xml, altrament no deixarà crear el document per evitar errors en el sistema. Un exemple correcte per a la creació d'un document xml seria:

<titol>novela0</titol>

<autor>Miguel de Cervantes</autor>

<contingut>Montaña</contingut>

o

<titol>novela0</titol>

<autor>Miguel de Cervantes</autor>

<contingut>Montaña</contingut>

Aquesta estructura especifica està a l'abast de l'usuari a l'hora de crear el document en la part superior de l'aplicació.

3.3.7 Encriptació de les dades d'usuari

Per conservar les dades d'usuari d'una sessió a una altra, en comptes de guardar les parelles <usuari,contrasenya> en un fitxer de text pla, un xml o un JSON per exemple, hem decidit que seria millor serialitzar-lo.

Ens hem decidit per l'opció de la serialització per intentar emular d'alguna forma, tot i que molt més simple clarament, la manera en la que es gestionarien aquestes dades en un projecte real, ja que les contrasenyes dels usuaris que fan servir el sistema són sensibles i no és convenient tenir-les guardades en un fitxer fàcilment accessible des de fora del nostre sistema.

El paper que juga la serialització en aquesta qüestió és la de transformar les parelles <usuari, contrasenyes> en cadenes de bytes ilegibles des de fora del sistema.

La serialització i des-serialització de les dades dels usuaris es realitzen en el moment al iniciar l'aplicació i en el moment del *login* respectivament, per tal de que estiguin actualitzats.

3.4 Millora de les ED i Algorismes per la segona entrega

3.4.1 Estructures de Dades

L'única estructura de dades relativament important és un HashMap amb les parelles <usuari,contrasenya> per tal de poder implementar el sistema d'usuaris. I el motiu pel que s'ha utilitzat aquesta *ED* és pel mateix motiu mencionat en apartats anteriors.

Pel que fa a la resta de les estructures de dades, no s'ha implementat cap canvi substancial en cap. L'estructura del fitxer invertit s'ha mantingut igual, i la del *file term vector* també, ja que per la primera entrega es van crear de forma que fossin fàcilment escalables, per evitar que al afegir noves funcionalitats fos necessari canviar les ED's.

3.4.2 Algorismes

Pel que fa a la resta d'algorismes de la primera entrega, no ha estat necessari modificar cap, doncs funcionaven correctament. Únicament ha estat necessari modificar el de la cerca booleana, doncs tenia alguns aspectes a millorar.

La funció auxiliar que ens permet detectar si l'expressió booleana creada per l'usuari està ben construïda en termes de paréntesis, *checkOpBool* l'hem fet més eficient convertint-la en una funció que implementa un Stack, *getBrackets*.

Les diferències en termes d'eficiència entre una implementació recursiva i una altra que utilitza un stack poden variar en funció de la situació específica en la qual s'utilitzin. En general, les implementacions recursives poden ser més fàcils de llegir i entendre que les implementacions que utilitzen un stack, ja que la recursivitat segueix un patró lògic i fàcil de seguir. No obstant això, en termes d'eficiència, les implementacions que utilitzen un stack solen ser més eficients, ja que el stack s'utilitza per a emmagatzemar temporalment les dades i les operacions, la qual cosa pot evitar haver de tornar a realitzar càlculs realitzats anteriorment.

Per a facilitar l'experiència de l'usuari en la cerca booleana hem afegit diferents funcionalitats d'ordenació en l'historial de les expressions booleanes, a més de les funcionalitats bàsiques afegides en aquesta segona entrega en el seu resultat, com hem comentat anteriorment en l'apartat 3.3.4.

A més, hem volgut millorar l'algoritme d'actualització de l'Inverted file, ja que és un algoritme molt costos, a més a més si tenim un nombre gran de documents. L'intenció principal d'aquesta millora, és intentar emular un sistema més realista i que s'apropa més a la realitat. Però per altra banda, és necessari mantenir una actualització constant d'aquesta representació interna ja que sinó perdem la precisió i no podem obtenir les cerques de similitud y de rellevància de documents amb tanta exactitud.

Per això, hem decidit actualitzar la representació cada n iteracions, sent 'n' una variable en funció el nombre actual de documents del sistema. Mentres n sigui menor a 10, actualitzarem l'Inverted file cada cop que es modifiqui un document, ja que al tenir pocs documents no serà tant costos renovar la representació i cada detall dels documents tindrà una major rellevància. Quan es trobi entre 10 i 30, les actualitzacions les realitzarem cada 3 modificacions d'un document. Finalment, si el conjunt de documents, conté una quantitat major a 30, l'actualització només es farà cada 5 canvis als documents.

Hem elegit aquests paràmetres perquè hem cregut que eren els més òptims per trobar un equilibri entre l'eficiència i la precisió del sistema.

Per aquesta segona entrega el que sí que s'ha fet és implementar la nova funcionalitat, la 4.3 a l'enunciat del projecte. Donades *p* paraules (denotades col·lectivament com a *query*), i un enter *k*, obtenir els k documents més rellevants (en quant a contingut) per aquesta query.

Els passos que es segueixen per aquest algorisme no són res complicats, gràcies a l'escalabilitat de les estructures de dades del fitxer invertit i el *file term vector*, que són les que principalment s'han utilitzat per aquesta funcionalitat.

1. Donada la *query* de *p* paraules, es construeix un *file term vector* com si fos un document. Per la construcció d'aquesta ED, com pels documents, es fa servir el fitxer invertit per tal de calcular-ne els pesos *tf-idf* de cadascuna de les paraules.
2. Un cop creat un objecte *fileTermData* per la *query*, es procedeix a calcular la similitud d'aquest amb el de la resta de documents.
3. Els resultats obtinguts s'ordenen de major a menor en base al grau de similitud
4. Es seleccionen únicament els *k* documents més semblants.

4. Secció tests

Hem realitzat un conjunt de tests, un per cada classe utilitzant el framework JUnit, fent un total de 7 tests. A més també existeix una classe Suite, la qual realitza el conjunt de tots els tests. L'objectiu d'aquests és comprovar el correcte funcionament de totes les classes.

Per executar el .bat que ens permet veure el resultat dels nostres jocs de prova. Hem d'anar a la carpeta /FONT/src i executar l'arxiu executarJUNITS.bat (fent proves, en alguns PCs surt un warning dient que l'arxiu no es conegut i pot comportar un risc pel sistema, simplement li donem a "run anyway").

A més també volem veure el funcionament de totes les operacions i funcions, independentment de les altres classes i les altres funcions. Tots els tests ens han retornat el resultat esperat, per tant, estem satisfets amb els resultats.

5. Relació de les classes implementades per cada membre de l'equip

(classificades per packages)

Albert Vidal

Controladors	**Domini**	**Presència**
Controlador.java ControladorPresencia.java ControladorUsuaris.java	ConjuntDocuments.java OpBoleana.java InvertedFile.java main.java	Totes
Exceptions	**Test**	
	InvertedFileTest.java wordDataTest.java	

Ignasi Juez

Controladors	**Domini**	**Presència**
Controlador.java Controladorpresencia.java	ConjuntDocuments.java InvertedFile.java OpBoleana.java	Totes menys tornarPuntRestauració.java registrarUsuari.java nouPuntRestauració.java gestorRestauracions.java eliminarUsuaris.java
Exceptions	**Test**	
	ConjuntDocumentTests.java	

Lluís LLull

Controladors	**Domini**	**Presència**
	InvertedFile.java ConjuntDocuments.java WordData.java	eliminaDoc.java
Exceptions	**Test**	
	ContingutTest.java DocumentTest.java fileTermDataTest.java InvertedFileTest.java wordDataTest.java ConjuntDocumentsTest.java	

Iván Serrano

Controladors	**Domini**	**Presència**
Controlador.java Controladorpresencia.java	Contingut.java Document.java fileTermData.java InvertedFile.java wordData.java ConjuntDocuments.java	ResultatCercaSimilars.java CercaSimilars.java menu.java login.java
Exceptions	**Test**	
autorNotFoundException.java fileNotFoundException.java fileAlreadyExisting.java		

6. Llibreries utilitzades

java.io

java.io.BufferedReader, java.io.File, java.io.FileReader, java.io.FileWriter, java.io.IOException, java.io.Serializable

java.nio.file

java.nio.file.Files, java.nio.file.Path, java.nio.file.Paths

java.util

java.util.HashMap, java.util.List, java.util.Map; java.util.TreeMap, java.util.Map.Entry

A moltes classes s'ha incluit java.util.* per tenir-les agrupades

java.awt

java.awt.Color, java.awt.Font, java.awt.event.ActionEvent, java.awt.event.ActionListener

javax.swing

javax.swing.filechooser.FileSystemView, javax.swing.JButton, javax.swing.JFrame, javax.swing.JLabel, javax.swing.JFileChooser, javax.swing.JTextField, javax.swing.filechooser.FileSystemView, javax.swing.JFileChooser, javax.swing.BorderFactory, javax.swing.JPasswordField, javax.swing.JComboBox

java.lang

java.lang.Math