Gestor de Archivos

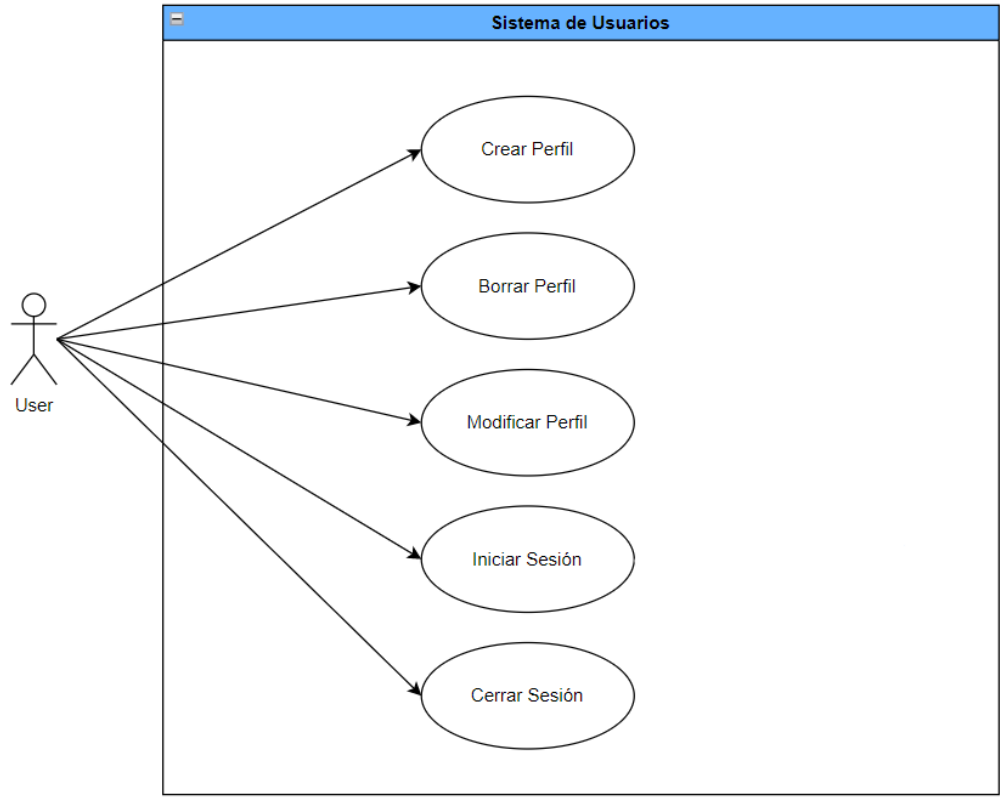
Grau PROP Q1 Curs 2022-2023
1ª entrega Documentación (version 1.0)

Miembros del grupo
Aranda Sánchez, Hugo
Gallardo Abellán, Alejandra
Ruiz Vives, Albert
Riera Puertas, Eric

ÍNDICE

1. CASOS DE USO

1.1 Sistema de Usuarios



Crear Perfil
Actor: Usuario
Precondición:

- El usuario no está registrado

Detonante: El usuario quiere tener un perfil para usar la aplicación en su máximo potencial
Escenario Principal:

1) El usuario proporciona unos parámetros (contraseña dos veces, nombre, etc.) de cuenta coherentes y compatibles con los que ya tenemos en la base de datos.
2) El sistema valida los valores y crea una cuenta con contraseña y nombre dados

Extensiones:

- Contraseña no coincide: el sistema informa del error. Vuelve al estado 1.
- Username existente o no compatible: el sistema informa del error. Vuelve al estado 1.
- Contraseña no reúne los criterios de calidad (contraseña vacía): el sistema informa del error. Vuelve al estado 1.

Borrar Perfil

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere darse de baja

Escenario Principal:

1) El sistema exige confirmación mediante inserción de contraseña.
2) El usuario inserta la contraseña
3) El sistema valida la contraseña y elimina del sistema la cuenta

Extensiones:

- Contraseña errónea: el sistema informa del error. Vuelve al estado 1.

Modificar Perfil

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere modificar alguno de los valores de su perfil

Escenario Principal:

1) El sistema exige la reinserción de contraseña antes de realizar cambios en el perfil como medida de seguridad
2) El sistema abre un menú en el cual se pueden modificar todos los valores relacionados con la cuenta
3) El usuario confirma esos cambios

Extensiones:

- Contraseña errónea: el sistema informa del error. Vuelve al estado 1
- Cambio de username por otro ya existente: el sistema informa del error. Vuelve al estado 1.

Iniciar Sesión

Actor: Usuario

Precondición:

- El usuario está registrado y no logueado

Detonante: El usuario quiere iniciar sesión en una cuenta existente

Escenario Principal:

1) El usuario pone sus credenciales de inicio de sesión (nombre de usuario y password)
2) Se valida la contraseña y usuario, si estos son correctos se le da acceso a la cuenta y el resto de la aplicación

Extensiones:

- Contraseña errónea: el sistema informa del error. Vuelve al estado 1
- El nombre no existe en el sistema: el sistema informa del error. Vuelve al estado 1

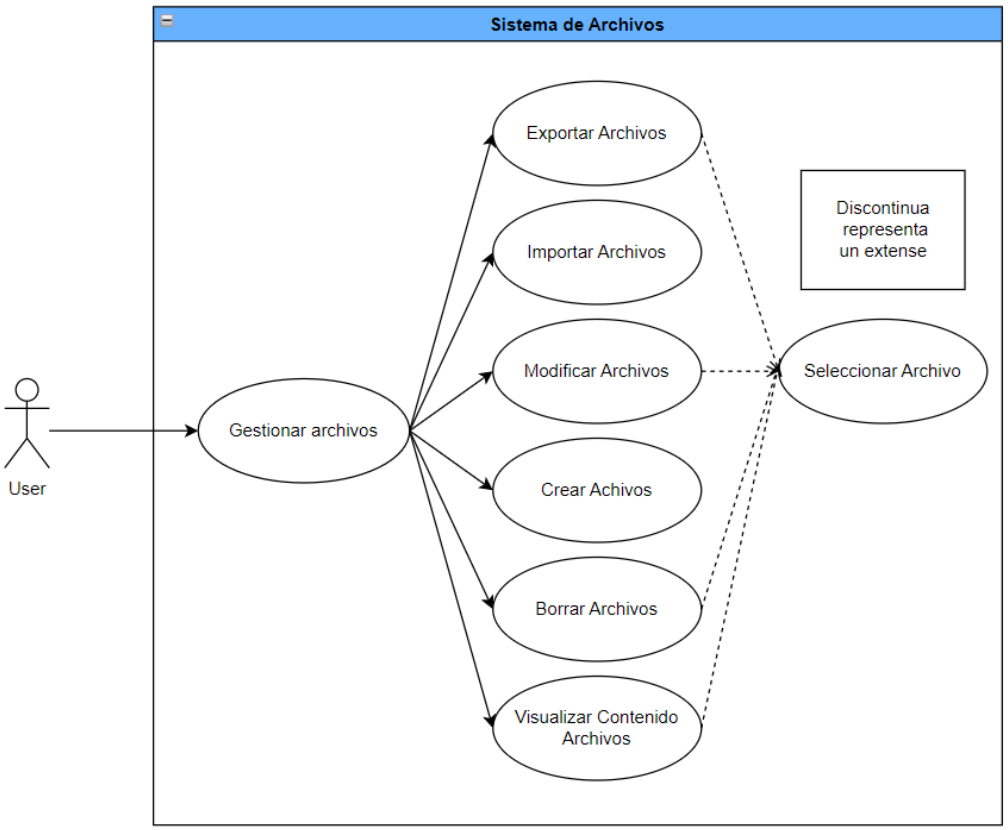
Cerrar Sesión
Actor: Usuario
Precondición: El usuario está registrado y logueado
Detonante: El usuario quiere cerrar sesión de una cuenta existente
Escenario Principal:

1) El usuario confirma el cierre de sesión con confirmación gráfica

Extensiones:

- Confirmación no aceptada: el sistema informa del error. Vuelve al estado 1

1.3 Sistema de Archivos



Exportar Archivos
Actor: Usuario
Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere cargar en el sistema uno o más ficheros.
Escenario Principal:

1) El usuario selecciona el archivo(s) que quiere exportar
2) El sistema carga el archivo(s) en el equipo del usuario

Extensiones:

- Archivo a cargar no existe: el sistema informa del error. Vuelve al estado 1.
- Archivo repetido (mismo nombre, autor): se ofrecerá 3 opciones:
    a) **Sobrescribir archivo** de su equipo con el nuevo
    b) **Cargar fichero** con otro título o autor
    c) Cancelar proceso

Info Adicional:

En caso que sea un conjunto, se analizará uno por uno si ya existe en el equipo del usuario un fichero con el mismo nombre

Importar Archivos

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere guardar en el sistema un archivo de su equipo

Escenario Principal:

1) El usuario selecciona el archivo o conjunto de archivos y selecciona la opción importar
2) El sistema validará el archivo y lo guardará en el sistema

Extensiones:

- Archivo a importar no existe: el sistema informa del error. Vuelve al estado 1.
- Archivo repetido (mismo título y autor): se ofrecerá 3 opciones:
    a) "**Sobrescribir archivo**" de su sistema con el nuevo. Salta al estado 3 de "**Modificar Archivo**"
    b) Guardar con otro título o autor
    c) Cancelar proceso
- Formato desconocido: el sistema informa del error. Vuelve al estado 1.

Modificar Archivos

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere modificar el título, autor o contenido del archivo.

Escenario Principal:

1) El usuario selecciona el archivo y selecciona la opción modificar
2) El sistema muestra una interfaz con título, autor y contenido del archivo
3) El usuario modifica cualquiera de los campos
4) El sistema valida el archivo y lo guarda

Extensiones:

- Archivo a modificar no existe: el sistema informa del error. Vuelve al estado 1.
- Archivo repetido (mismo título y autor): se proponen 3 opciones:
    a) **Sobrescribir archivo** con ese nombre. Vuelve al estado 3
    b) **Introducir título o autor** nuevo
    c) Cancelar proceso

Info Adicional:

- Solo se permite seleccionar un archivo a la vez

Crear Archivos

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere crear un archivo(s) del sistema

Escenario Principal:

1) El usuario selecciona la opción de crear archivo
2) El sistema muestra una interfaz con título, autor y contenido vacío.
3) El usuario rellena los campos
4) El sistema valida el archivo y lo guarda

Extensiones:

- Archivo ya existe (mismo título y autor): se ofrecerá 3 opciones:
    a) "**Sobrescribir archivo**" de su sistema con el nuevo. Vuelve al estado 3
    b) Guardar con otro título o autor
    c) Cancelar proceso
- Formato desconocido: el sistema informa del error. Vuelve al estado 1
- Archivo ya existente en el sistema: el sistema informa del error. Vuelve al estado 1.

Borrar Archivos

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere borrar un archivo(s) del sistema

Escenario Principal:

1) El usuario selecciona el archivo o conjunto de archivos y selecciona la opción borrar
2) El sistema borra el archivo

Extensiones:

- Archivo a eliminar no existe: el sistema informa del error. Vuelve al estado 1.
- El archivo indicado no existe: el sistema informa del error. Vuelve al estado 1.

Visualizar contenido Archivos

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario ha buscado un archivo(s) en el sistema del cual quiere obtener su contenido

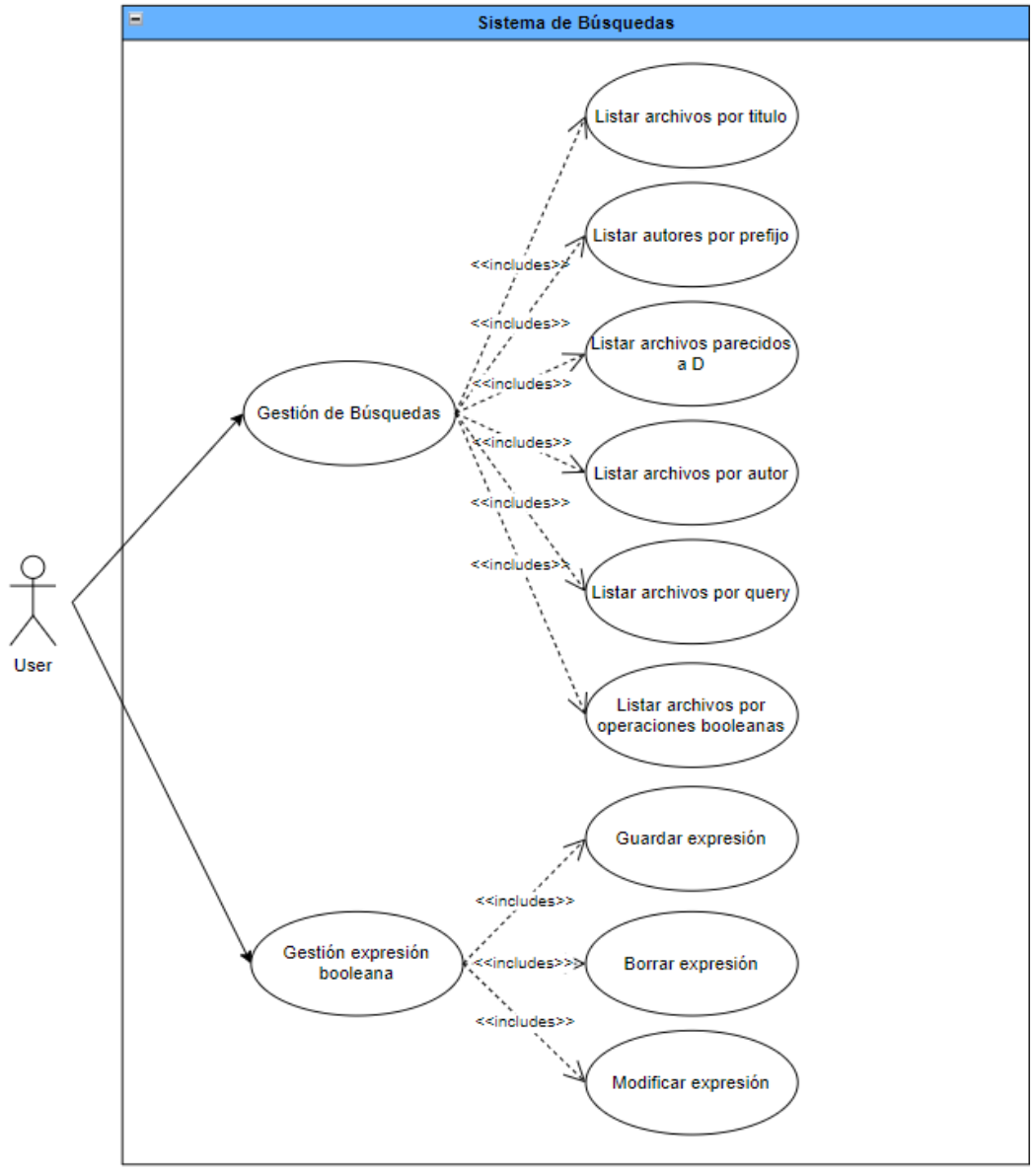
Escenario Principal:

1) El usuario selecciona el archivo y elige la opción de "extraer contenido"
2) El sistema abrirá una pestaña de visualización del contenido

Extensiones:

- Archivo a eliminar no existe: el sistema informa del error. Vuelve al estado 1.

1.3 Sistema de búsqueda



Listar archivos por título
Actor: Usuario
Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere realizar una búsqueda de un archivo por su título
Escenario principal:

1) El usuario introduce el nombre del título a buscar
2) El sistema muestra una lista con los archivos que contienen la información introducida en el título

Extensiones: —
Info Adicional:

- No hay archivos en el listado: mostrar lista vacía con mensaje especial

Listar autores por prefijo

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere realizar una búsqueda de un autor por un prefijo

Escenario principal:

1) El usuario introduce un prefijo
2) El sistema muestra una lista con los autores que empiezan por el prefijo introducido

Extensiones: —

Info Adicional:

- No hay autores: mostrar lista vacía con mensaje especial

Listar archivos parecidos a D

Actor: Usuario

Precondición:

- El usuario está registrado y logueado
- k > 0

Detonante: El usuario quiere realizar una búsqueda de k archivos parecidos a D

Escenario principal:

1) El usuario informa de un archivo D y un natural k
2) El sistema muestra una lista con los k archivos más parecidos a D

Extensiones:

- Archivo D no existe: el sistema informa del error. : el sistema informa del error. Vuelve al estado 1.

Listar archivos por autor

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere realizar una búsqueda de los archivos de un autor

Escenario principal:

1) El usuario introduce el nombre de un autor
2) El sistema muestra una lista con los archivos que son del autor introducido

Extensiones: —

Info Adicional:

- No hay archivos en el listado: mostrar lista vacía con mensaje especial

Listar archivos por query

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere realizar una búsqueda de los archivos más relevantes dadas p palabras

Escenario Principal:

1) El usuario introduce p palabras(query) y un natural k
2) El sistema muestra una lista con los archivos más relevantes para esta query

Extensiones: —

Info Adicional:

- Query no válida (mala parentización, comillas…): el sistema informa del error. Vuelve al estado 1.
- No hay archivos en el listado: mostrar lista vacía con mensaje especial

Listar por Expresión booleana

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere realizar una búsqueda por operaciones booleanas

Escenario Principal:

1) El usuario introduce una expresión booleana en el gestor de búsqueda
2) El sistema valida que la expresión sea correcta. Si lo es, el sistema guarda la expresión para usos futuros (salta al caso de uso **Guardar Expresión**)
3) El sistema devuelve un listado con todos los ficheros que cumplen las condiciones especificadas por la expresión

Extensiones:

- Expresión booleana no válida: el sistema informa del error y descarta la nueva expresión. Vuelve al estado 1.

Info adicional: Si no hay archivos que cumplan la expresión, devuelve una lista vacía con un mensaje especial de “no hay resultados en la búsqueda”.

Guardar Expresión

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario ha realizado una búsqueda por operaciones booleanas

Escenario Principal:

1) El sistema guarda la expresión usada si ésta era válida

Extensiones: —

Borrar Expresión

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere eliminar una expresión booleana guardada en el sistema

Escenario Principal:

1) El usuario selecciona una expresión y elige la opción de borrar
2) El sistema elimina la expresión

Extensiones:

- La expresión no existe: el sistema informa del error. Vuelve al estado 1.

Modificar Expresión

Actor: Usuario

Precondición:

- El usuario está registrado y logueado

Detonante: El usuario quiere modificar una expresión booleana guardada en el sistema

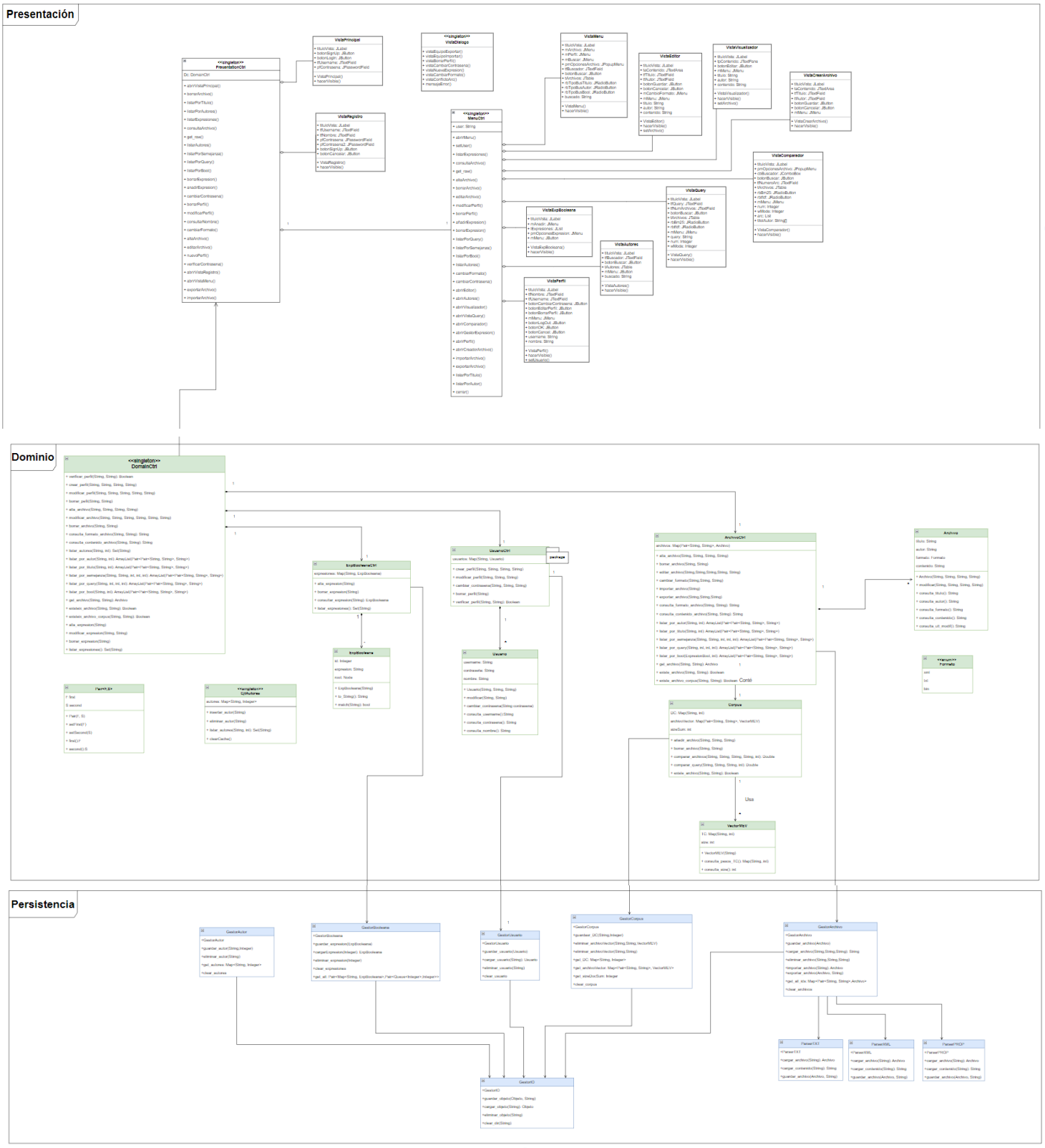
Escenario Principal:

1) El usuario selecciona una expresión y elige la opción de modificar
2) El sistema pide que se hagan las modificaciones pertinentes (añadir nueva expresión para sustituir)
3) El usuario introduce la nueva expresión, el sistema la valida y guarda la modificación

Extensiones:

- La expresión booleana seleccionada no existe: el sistema informa del error. Vuelve al estado 1.
- Nueva expresión booleana no válida (mala parentización, malas comillas…): el sistema informa del error. Vuelve al estado 1 y se cancela el proceso.

2. DIAGRAMA DE CLASES



Link del diagrama:
https://drive.google.com/file/d/1hf7QAvPbvfkI6KvcI1OdRJsKe6kn_GoM/view?usp=sharing

NOTA: Todas las agregaciones de la capa de presentación son 1-1.

2.1 Capa de Presentación

2.1.1 Descripción de clases

Vistas:

VistaPrincipal

Vista del inicio de sesión en el sistema. Se encarga de dar acceso al menú principal si se introducen las credenciales correctas (nombre de usuario y contraseña). Contiene una opción de registro en caso que no se disponga de un perfil creado.

Cardinalidad: 1

Atributos:
- tituloVista
- botonLogIn
- tfUsername
- pfContrasena
- botonSignUp

VistaRegistro

Vista de creación de cuenta. Se encarga de registrar nuevas credenciales para dar de alta en el sistema una nueva cuenta con los datos introducidos. Necesita llenar un segundo campo de confirmación de contraseña (tfContrasena2) para crear el perfil.

Cardinalidad: 1

Atributos:
- tituloVista
- tfUsername
- tfNombre
- pfContrasena
- pfContrasena2
- botonSignUp
- botonCancelar

VistaMenu

Menú principal del sistema. Contiene un buscador con un listado de todos los archivos existentes en el sistema, identificados por título y autor. Muestra una barra de menú en la cual ofrece varias interacciones como gestionar el perfil, archivos y expresiones booleanas del sistema o distintos tipos de buscador.

Si se selecciona un archivo del listado, se muestra un menú contextual con las opciones posibles de interacción con el archivo como editar, borrar, visualizar o exportar.

Cardinalidad: 1

Atributos:
- tituloVista
- mArchivo
- mPerfil
- mBuscar
- pmOpcionesArchivo
- tfBuscador
- botonBuscar
- tArchivos
- rbTipoBusTitulo
- rbTipoBusAutor
- rbTipoBusBool
- buscado
    - ➔ Valor inicial: String vacía

VistaPerfil

Vista de gestión de la cuenta iniciada. Contiene una visualización del usuario y el nombre, junto con las opciones de modificar estos o la contraseña. Esta vista también dispone la opción de borrar perfil, el cual te redirige a la vistaPrincipal una vez se confirma la eliminación.

Cardinalidad: 1

Atributos:
- tituloVista
- tfNombre
- tfUsername
- botonCambiarContrasena
- botonEditarPerfil
- botonBorrarPerfil
- mMenu
- botonLogOut
- botonOK
- botonCancel
- username
    - ➔ Valor inicial: String vacía
- nombre
    - ➔ Valor inicial: String vacía

VistaCrearArchivo

Vista de crear archivo. Se encarga de crear un archivo con el título, autor y contenido indicados. Contiene una opción de guardar y otra para cancelar los cambios.

Cardinalidad: 1

Atributos:

- tituloVista
- taContenido
- tfTitulo
- tfAutor
- botonGuardar
- botonCancelar
- mMenu

VistaVisualizador

Visualizador de archivo. Se encarga de mostrar el título, autor y contenido del archivo seleccionado en el menú principal. Contiene una opción de edición del archivo.

Cardinalidad: 1

Atributos:
- tituloVista
- tpContenido
- botonEditar
- mMenu
- titulo
    - ➔ Valor inicial: String vacía
- autor
    - ➔ Valor inicial: String vacía
- contenido
    - ➔ Valor inicial: String vacía

VistaEditor

Editor de archivo. Se encarga de mostrar y dar la posibilidad de editar el título, autor y contenido del archivo seleccionado en el menú principal. Contiene una opción de guardar y otra para cancelar los cambios.

Cardinalidad: 1

Atributos:
- tituloVista
- taContenido
- tfTitulo
- tfAutor
- botonGuardar
- botonCancelar
- mCambioFormato
- mMenu
- titulo
    - ➔ Valor inicial: String vacía
- autor
    - ➔ Valor inicial: String vacía
- contenido

➔ Valor inicial: String vacía

VistaComparador

Vista de buscador por semejanza de archivos. Contiene un buscador con el cual se puede seleccionar un archivo y el número de archivos parecidos al seleccionado que queremos que aparezcan en el listado, identificados por título y autor.

Si se selecciona un archivo del listado, se muestra un menú contextual con las opciones posibles de interacción con el archivo como editar, borrar, visualizar o exportar.

Cardinalidad: 1

Atributos:
- tituloVista
- pmOpcionesArchivo
- cbBuscador
- botonBuscar
- tfNumeroArc
- tArchivos
- rbBm25
- rbtfdf
- mMenu
- num
    ➔ Valor inicial: 0
- wMode
    ➔ Valor inicial: 0
- arc
    ➔ Valor inicial: Lista vacía
- titolAutor
    ➔ Valor inicial: String[] de tamaño 2 vacía

VistaQuery

Vista de buscador por una query. Contiene un buscador con el cual se introduce la query y el número archivos que contienen la query queremos que aparezcan en un listado, identificados por título y autor.

Si se selecciona un archivo del listado, se muestra un menú contextual con las opciones posibles de interacción con el archivo como editar, borrar, visualizar o exportar.

Cardinalidad: 1

Atributos:
- tituloVista
- tfQuery
- botonBuscar
- tArchivos

- rbBm25
- rbtfdf
- tfNumArchivos
- mMenu
- num
    - ➔ Valor inicial: 0
- wMode
    - ➔ Valor inicial: 0
- query
    - ➔ Valor inicial: String vacía

VistaAutores

Buscador de autores. Se encarga de las búsquedas de autores, es decir, la vista consiste en un buscador junto con un listado de autores existentes del sistema.

Cardinalidad: 1

Atributos:
- tituloVista
- tfBuscador
- botonBuscar
- tAutores
- mMenu
- buscado
    - ➔ Valor inicial: String vacía

VistaExpBooleana

Vista de gestión de expresiones booleanas. Consiste en un listado de las expresiones booleanas que ha creado el usuario, que incluye un menú contextual de opciones de la expresión booleana, como editar o borrar. Además la vista incluye la opción de añadir una nueva expresión booleana.

Cardinalidad: 1

Atributos:
- tituloVista
- mAnadir
- lExpresiones
- pmOpcionesExpresion
- mMenu

VistaDialogo

Vista de diálogo. Esta clase muestra distintas vistas de diálogo según el tipo de diálogo requerido. Incluyen por ejemplo vistas donde se solicitan inputs, escoger una opción o donde se muestran mensajes de error.

Cardinalidad: 1..*

Atributos: —
Nota: Los atributos como los botones de opción se generan automáticamente por el JOptionPane.

Controladores:

MenuCtrl

Se encarga de la gestión de todas las vistas que componen el sistema una vez iniciada la sesión, además de la comunicación con el controlador de presentación para obtener los valores necesarios para llevar a cabo las funcionalidades. Tiene presente si un usuario ha iniciado sesión y cuál es.

Cardinalidad: 1

Atributos:
- user
  - ➔ Valor inicial: String vacía

PresentationCtrl

Se encarga de la gestión de la capa de presentación. Es la responsable de la comunicación entre las vistas de registro y principal y el MenuCtrl. Se comunica con la capa de dominio a través de DomainCtrl para obtener los datos necesarios.

Cardinalidad: 1

Atributos: —

2.2 Capa de Dominio

2.2.1 Restriccions Textuals

Claus Primàries

(**Usuario**, username); (**Archivo**, título+autor); (**ExpBooleana**, expresion)

2.2.2 Descripción de Clases

Clases:

Usuario

Contiene todos los datos de un usuario registrado. Se encarga de la gestión de información del usuario, como por ejemplo su registro, modificación o consultas.

Cardinalidad: 1..*

Atributos:

- username
    ➜ Valor inicial: valor entrado
- contraseña
    ➜ Valor inicial: valor entrado
- nombre: nombre del usuario
    ➜ Valor inicial: valor entrado

Archivo

Contiene todas las características de un archivo. Se divide en distintos formatos: lenguaje de marcado extensible (xml), texto plano (txt) o binario (bin). Se encarga de las consultas de información de un archivo en concreto y de la modificaciones de estos.

Cardinalidad: 1..*

Atributos:

- título
    ➜ Valor inicial: valor entrado

- autor
    ➜ Valor inicial: valor entrado
- formato
    ➜ Valor inicial: valor entrado
- contenido
    ➜ Valor inicial: valor entrado

Corpus

Guarda el número de documentos que contienen cada palabra del Corpus (denominado *dc*). Además, se encarga de la comparación de contenido de archivos en Modelo de Espacio Vectorial del sistema y la evaluación de pesos.

Cardinalidad: 1

Atributos:

- DC
    ➜ Valor inicial: map vacío

- archivoVector
    ➜ Valor inicial: map vacío

- SizeSum
    ➜ Valor inicial: 0

VectorMEV

Almacena el número de apariciones de una palabra dentro del documento (denominado *tc*). Además, se encarga del cálculo y la consulta de pesos *tc* de un texto entrado que corresponde al contenido de un archivo.

Cardinalidad: *

Atributos:
- TC
    - ➜ Valor inicial: variable según el texto entrado
- size
    - ➜ Valor inicial: suma de los valores del TC

ExpBooleana
Contiene la información necesaria para poder evaluar textos de acuerdo a una expresión booleana que la define.

Cardinalidad: *

Atributos:
- id
    - ➜ Valor inicial: identificador dado
- expresion
    - ➜ Valor inicial: texto de entrada
- root
    - ➜ Valor inicial: variable según el texto de entrada

CjtAutores (Singleton)
Guarda eficientemente todos los autores junto al número de Archivos de los cuales es el autor. Se encarga de la gestión de consultas de los autores del sistema y de listarlos.

Cardinalidad: 1

Atributos:
- autores
    - ➜ Valor inicial: map persistido

Controladores

ArchivoCtrl
Contiene todas las instancias de Archivo eficientemente. Se encarga de gestionar todas las interacciones con archivos como consultas, modificaciones, registro, etc.

Cardinalidad: 1

Atributos:
- archivos
    - ➜ Valor inicial: map persistido

UsuarioCtrl
Guarda eficientemente todos los usuarios registrados del sistema. Se encarga de los registros, verificaciones, modificaciones y borrados de perfil.

Cardinalidad: 1

Atributos:
- usuarios
    ➜ Valor inicial: map vacío

ExpBooleanaCtrl
Contiene todas las expresiones booleanas guardadas por el usuario. Este controlador es responsable de la gestión de expresiones booleanas, como por ejemplo: cargar, guardar, borrar y consultar.

Cardinalidad: 1

Atributos:
- expresiones
    ➜ Valor inicial: map persistido
- max_id
    ➜ Valor inicial: identificador máximo
- disp
    ➜ Valor inicial: queue persistido

DomainCtrl
Responsable de la gestión del dominio y de la comunicación de la capa de presentación con todos los controladores del dominio.

Asociaciones

ArchivoCtrl - Corpus (1 - 1)
El controlador de archivos accede a corpus para realizar los cálculos de pesos de los contenidos de los archivos.

Corpus - VectorMEV (1 - *)
El corpus guarda las relaciones entre los archivos y los vectores MEV y accede a ellos para realizar los cálculos de los vectores tf y df.

ArchivoCtrl - GestorArchivo (1 - 1)
El controlador de archivos accede a su gestor para persistir sus datos en disco.

ExpBooelanaCtrl - GestorBooleana (1 - 1)
El controlador de expresion booleana accede a su gestor para persistir sus datos en disco.

Corpus - GestorCorpus (1 - 1)
Corpus accede a su gestor para persistir sus datos en disco.

UsuarioCtrl - GestorUsuario (1 - 1)
El controlador de usuarios accede a su gestor para persistir sus datos en disco.

Agregaciones

Usuario - UsuarioCtrl (* - 1)
UsuarioCtrl opera sobre el conjunto de instancias de Usuario del sistema.

ArchivoCtrl - Archivo (1 - *)
ArchivoCtrl opera sobre el conjunto de instancias de Archivo del sistema.

ExpBooleanaCtrl - ExpBooleana (1 - *)
ExpBooleanaCtrl opera sobre el conjunto de instancias de ExpBooleana del sistema.

DomainCtrl - ExpBooleanaCtrl (1 - 1)
DomainCtrl opera sobre ExpBooleanaCtrl para obtener la información pedida por la capa de presentación.

DomainCtrl - UsuarioCtrl (1 - 1)
DomainCtrl opera sobre UsuarioCtrl para obtener la información pedida por la capa de presentación.

DomainCtrl - ArchivoCtrl (1 - 1)
DomainCtrl opera sobre ArchivoCtrl para obtener la información pedida por la capa de presentación.

2.3 Capa de Persistencia

2.3.1 Descripción de Clases

Gestores:

GestorUsuario

Gestiona la persistencia de las instancias de Usuario. Permite guardar (persistir), eliminar y cargar los datos individualmente “on demand”.

Cardinalidad: 1

Atributos:

- path
  - ➜ Valor inicial: “Path” del directorio donde se guardan los datos persistidos del gestor.

GestorAutores

Gestiona la persistencia del mapa “autores” de la clase CjtAutores. Permite guardar y eliminar entradas individuales del mapa. Así como cargar el mapa persistido entero “on boot”.

Cardinalidad: 1

Atributos:

- path
  - ➜ Valor inicial: “Path” del directorio donde se guardan los datos persistidos del gestor.

GestorCorpus

Gestiona la persistencia de los mapas “archivoVector” y “DC” de la clase Corpus. Así como el entero “sizeDocSum”. Permite guardar y eliminar entradas individuales de los mapas y el entero. Así como cargar los mapas persistidos enteros y el entero “on boot”.

Cardinalidad: 1

Atributos:

- path
  - ➜ Valor inicial: “Path” del directorio donde se guardan los datos persistidos del gestor.

GestorBooleana

Gestiona la persistencia de las instancias de ExpBooleana. Permite guardar (persistir), eliminar y cargar los datos enteros “on boot”.

Cardinalidad: 1

Atributos:

- path
  - ➜ Valor inicial: “Path” del directorio donde se guardan los datos persistidos del gestor.

GestorArchivo

Gestiona la persistencia de las instancias de Archivos. Permite guardar (persistir), eliminar y cargar ciertos datos al iniciar “on boot”, tales como el titulo, autor y formato. Además de poder consultar el contenido completo de un archivo particular “on demand”. Guarda los archivos completos en disco en su formato indicado para permitir consultas externas.

Cardinalidad: 1

Atributos:

- path

- ➜ Valor inicial: “Path” del directorio donde se guardan los datos persistidos del gestor.

GestorIO

Gestiona la lectura y escritura de los archivos persistidos. Permite guardar un objeto en persistencia a partir de un “path” y un objeto cualquiera y cargar un objeto a partir de su “path”.

Cardinalidad: 1

Atributos:

ParserTXT

Clase encargada del tratado de escritura y lectura de archivos del tipo “.txt”.

Cardinalidad: 1

Atributos: —

ParserXML

Clase encargada del tratado de escritura y lectura de archivos del tipo “.xml”.

Cardinalidad: 1

Atributos: —

ParserPROP

Clase encargada del tratado de escritura y lectura de archivos del tipo “.prop”.

Cardinalidad: 1

Atributos: —

4. MÉTODOS Y ATRIBUTOS

4.1 Archivo

Atributos

título:
String que junto a autor hace de clave externa para la clase.
autor:
String que junto a título hace de clave externa para la clase.
formato:
Enumerador que indica el formato del archivo almacenado.
contenido:
String que contiene el contenido del archivo

Métodos

Archivo(String titulo, String autor, String formato, String content):
Crea una instancia de la clase Archivo con título, autor, formato y contenido como los parámetros string pasados.

void modificar(String titulo, String autor, String formato, String content):
El archivo actualiza sus atributos con los parámetros correspondientes.

4.2 ArchivoCtrl

Atributos

archivos:
Map<Pair<String, String>,Archivo> que contiene todos los archivos almacenados según su clave primaria.

Métodos

ArchivoCtrl():
Crea una instancia de la clase Archivo Control con el map inicializado según el estado de la persistencia en disco.

void alta_archivo(String titulo, String autor, String formato, String content):
Se da de alta un archivo en el sistema con los datos dados por los parámetros. También se añade en persistencia de forma dinámica una representación de este archivo en su formato indicado.

void borrar_archivo(String titulo, String autor):
Se borra un archivo con la clave primaria

void editar_archivo(String titulo_ant, String autor_ant, String titulo, String autor, String contenido):
Esa función cambia los atributos del archivo con clave primaria titulo_ant y autor_ant por los nuevos título, autor y contenido dados. Sin cambiar el formato del mismo.

void cambiar_formato(String titulo, String autor, String new_formato):
Esa función cambia el formato del archivo con clave primaria título y autor al formato dado por el parámetro new_formato.

void importar_archivo(String path):
Importa el Archivo guardado en la dirección de disco indicada por el parámetro de entrada “path”. Verificando que su formato y contenido sean correctos.

void exportar_archivo(String titulo, String autor, String path):
Exporta el Archivo identificado por la clave externa “titulo” y “autor” a la dirección de disco indicada por el parámetro de entrada “path”.

String consulta_formato_archivo(String titulo, String autor):
Devuelve el formato del archivo con claves primarias titulo y autor

String consulta_contenido_archivo(String titulo, String autor):
Devuelve el contenido del archivo con claves primarias titulo y autor

String ArrayList<Pair<String, String>> listar_por_autor(String autor, int sortMode):
Esta función lista todos los archivos que tengan el parámetro autor como autor, esta lista contiene la clave primaria ordenada según unos modos de ordenación. El código de sortMode ordena lexicográficamente según estos códigos: 0 título creciente, 1 título decreciente, 2 autor creciente y 3 autor decreciente.

ArrayList<Pair<String, String>> listar_por_titulo(String titulo, int sortMode):
Esta función lista todos los archivos que tengan el parámetro título como título, como antes la lista contiene las claves primarias y tiene modos de ordenación.

ArrayList<Pair<String, String>> listar_por_semejanza(String titulo, String autor, int n, int weightMode, int sortMode):
Esta función lista n archivos parecidos al archivo identificado por el título y autor entrados, como antes la lista contiene las claves primarias y tiene modos de ordenación.

ArrayList<Pair<String, String>> listar_por_query(String query, int k, int weightMode, int sortMode):
Esta función lista k archivos parecidos a la query proporcionada, como antes la lista contiene las claves primarias y tiene modos de ordenación.

ArrayList<Pair<String, String>> listar_por_bool(ExpBooleana expresion, int sortMode):
Esta función lista los archivos que tienen una frase que cumple la expresión entrada, como antes la lista contiene las claves primarias y tiene modos de ordenación.

Archivo get_archivo(String titulo, String autor):
Retorna un archivo con claves título y autor.

boolean existe_archivo(String titulo, String autor):
Comprueba si el archivo con claves título y autor existe. Si existe retorna true, en caso contrario devuelve false.

boolean existe_archivo_corpus(String titulo, String autor):
Comprueba que el archivo identificado por la claves título y autor están registrados en el corpus.

4.3 CjtAutores

Atributos

autores:
Es un Map<String, Integer> que contiene todos los autores asociados al número de archivos donde aparece.

Métodos

CjtAutores():
Crea una instancia de la clase Conjunto Autores teniendo en cuenta la persistencia de sus datos.

void insertar_autor(String autor):
Registra otra aparición de autor en el conjunto.

void eliminar_autor(String autor):
Elimina una aparición de autor en el conjunto.

Set<String> listar_autores(String prefijo, int sortMode):
Devuelve un set con los strings de todos los autores de dentro del sistema según un sortMode: 0 ascendente y 1 descendente.

void clearCache():
Vacía el map autores.

4.4 DomainCtrl

Métodos

DomainCtrl():
Crea una instancia de la clase DomainCtrl que inicializa los otros controladores del dominio.

boolean verificar_perfil(String username, String contrasena):
Comprueba que la instancia de Usuario con clave externa “username” exista y tenga el atributo contrasena con el mismo valor que el parámetro de entrada “contrasena”. Devuelve cierto si se cumple la condición, falso de lo contrario. Sirve para hacer inicios de sesión.

void crear_perfil(String username, String contrasena1, String contrasena2 , String nombre):
Crea una nueva instancia de Usuario según los parámetros de entrada y la añade al mapa de usuarios. Se verifica que se hayan puesto los parámetros correctamente y que las 2 contraseñas sean iguales.

void modificar_perfil(String username_ant, String username, String nombre):
Modifica los atributos “username” y “nombre” del usuario con clave externa “username_ant” según los parámetros de entrada. Se verifica que se hayan puesto los parámetros correctamente.

void cambiar_contrasena(String username, String contrasena, String nueva_contrasena):
Cambia el atributo “contrasena” del usuario con clave externa “username” y contraseña “contrasena” según el parámetro de entrada. Se verifica que se haya puesto el parámetro correctamente.

void borrar_perfil(String username):
Elimina la instancia de Usuario con clave externa “username”

void alta_archivo(String titulo, String autor, String formato, String content):
Se da de alta un archivo en el sistema con los datos dados por los parámetros. También se añade en persistencia de forma dinámica una representación de este archivo en su formato indicado.

void borrar_archivo(String titulo, String autor):
Se borra un archivo con la clave primaria

void editar_archivo(String titulo_ant, String autor_ant, String titulo, String autor, String contenido):
Esa función cambia los atributos del archivo con clave primaria titulo_ant y autor_ant por los nuevos título, autor y contenido dados. Sin cambiar el formato del mismo.

void cambiar_formato(String titulo, String autor, String new_formato):
Esa función cambia el formato del archivo con clave primaria título y autor al formato dado por el parámetro new_formato.

void importar_archivo(String path):
Importa el Archivo guardado en la dirección de disco indicada por el parámetro de entrada “path”. Verificando que su formato y contenido sean correctos.

void exportar_archivo(String titulo, String autor, String path):
Exporta el Archivo identificado por la clave externa “titulo” y “autor” a la dirección de disco indicada por el parámetro de entrada “path”.

String consulta_formato_archivo(String titulo, String autor):
Devuelve el formato del archivo con claves primarias titulo y autor

String consulta_contenido_archivo(String titulo, String autor):
Devuelve el contenido del archivo con claves primarias titulo y autor

Set<String> listar_autores(String prefijo, int sortMode):
Devuelve un set con los strings de todos los autores de dentro del sistema según un sortMode: 0 ascendente y 1 descendente.

String ArrayList<Pair<String, String>> listar_por_autor(String autor, int sortMode):
Esta función lista todos los archivos que tengan el parámetro autor como autor, esta lista contiene la clave primaria ordenada según unos modos de ordenación. El código de sortMode ordena lexicográficamente según estos códigos: 0 título creciente, 1 título decreciente, 2 autor creciente y 3 autor decreciente.

ArrayList<Pair<String, String>> listar_por_titulo(String titulo, int sortMode):
Esta función lista todos los archivos que tengan el parámetro título como título, como antes la lista contiene las claves primarias y tiene modos de ordenación.

ArrayList<Pair<String, String>> listar_por_semejanza(String titulo, String autor, int n, int weightMode, int sortMode):
Esta función lista n archivos parecidos al archivo identificado por el título y autor entrados, como antes la lista contiene las claves primarias y tiene modos de ordenación.

ArrayList<Pair<String, String>> listar_por_query(String query, int k, int weightMode, int sortMode):
Esta función lista k archivos parecidos a la query proporcionada, como antes la lista contiene las claves primarias y tiene modos de ordenación.

ArrayList<Pair<String, String>> listar_por_bool(ExpBooleana expresion, int sortMode):
Esta función lista los archivos que tienen una frase que cumple la expresión entrada, como antes la lista contiene las claves primarias y tiene modos de ordenación.

Archivo get_archivo(String titulo, String autor):
Retorna un archivo con claves título y autor.

boolean existe_archivo(String titulo, String autor):
Comprueba si el archivo con claves título y autor existe. Si existe retorna true, en caso contrario devuelve false.

boolean existe_archivo_corpus(String titulo, String autor):
Comprueba que el archivo identificado por la claves título y autor están registrados en el corpus.

void alta_expresion(String exp):
Da de alta una expresión con clave primaria exp en el controlador si esta no existe ya. Además se le asigna un id único y consistente con los ya existentes asegurando la no fragmentación. Además se guarda en persistencia dinámicamente.

void borrar_expresion(String exp):
Borra una expresión con clave primaria exp en el controlador si esta existe ya. También se libera se id único para asegurar que no haya fragmentación y asignar este a una futura expresión. Además se borra de persistencia dinámicamente.

void modificar_expresion(String expresion_ant, String expresion):
Modifica la expresión con clave externa “expresion_ant” por la nueva expresión designada por el parámetro de entrada “expresion”.

Set<String> listar_expresiones():
Devuelve un set con los strings de todas las expresiones de dentro del sistema.

void clearCache():
Vacía el map autores.

4.5 ExpBooleana

Atributos

id:
Es un metadato que sirve de índice y es de utilidad a la hora de persistir los datos.
expStr:
Es un String que contiene la expresión en cuestión. No tiene un uso particular a nivel algorítmico más que mostrar la expresión si se pide.

root:
Es un elemento de la clase privada Nodo el cual contiene el árbol binario que representa la expresión booleana necesaria para la búsqueda.

Métodos

ExpBooleana(String exp, Integer identifier):
Crea una instancia de la clase Expresión con id identifier y expStr exp asegurando que la propia expresión dada por el string es adecuada según las normas establecidas por el sistema de búsqueda. Genera el árbol binario que representa la expresión.

boolean match(String str_eval):
Devuelve true o false según si el parámetro pasado cumple o no las condiciones establecidas por la expresión booleana.

4.6 ExpBooleanaCtrl

Atributos

expresiones:
Es un Map<String, ExpBooleana> que contiene todas las expresiones del controlador almacenadas según su clave primaria.
max_id:
Este representa el índice por asignar más alto en la cola de disponibles, es de utilidad a la hora de hacer que esta funcione correctamente. Ya que si eliminamos el más grande y no lo reponemos se rompe el sistema de índices.
disp:
Es una cola de integers la cual contiene los identificadores disponibles que debemos asignar a las nuevas expresiones creadas. Esta evita que suceda fragmentación en los identificadores.

Métodos

ExpBooleanaCtrl():
Crea una instancia de la clase Expresión Controlador teniendo en cuenta la persistencia de sus archivos.

void alta_expresion(String exp):
Da de alta una expresión con clave primaria exp en el controlador si esta no existe ya. Además se le asigna un id único y consistente con los ya existentes asegurando la no fragmentación. Además se guarda en persistencia dinámicamente.

void borrar_expresion(String exp):
Borra una expresión con clave primaria exp en el controlador si esta existe ya. También se libera se id único para asegurar que no haya fragmentación y asignar este a una futura expresión. Además se borra de persistencia dinámicamente.

Set<String> listar_expresiones():
Devuelve un set con los strings de todas las expresiones de dentro del sistema.

void ExpBooleana consultar_expresion(String exp):
Devuelve una expresión particular de clave externa exp.

4.7 Usuario

Atributos

username:
Es un String que contiene el nombre de usuario del perfil en cuestión. Hace de clave externa para la instancia de la clase.
contraseña:
Es un String que contiene la contraseña del perfil en cuestión. Permite hacer inicios de sesión junto a “username”
nombre:
Es un String que contiene el nombre real de la persona que usa el perfil en cuestión.

Métodos

Usuario(String username, String contrasena, String nombre):
Crea una instancia de la clase Usuario tomando el valor de los parámetros de entrada para la asignación de atributos.

void modificar(String username, String nombre):
Modifica los atributos “username” y “nombre” según los parámetros de entrada. En caso de poner un parámetro a “null”, ese atributo no sufrirá cambios.

void cambiar_contrasena(String contrasena):
Modifica el atributo “contrasena” según el parámetro de entrada. En caso de poner el parámetro a “null”, ese atributo no sufrirá cambios

4.8 UsuarioCtrl

Atributos

usuarios:
Es un Map<String, Usuario> que contiene los usuarios del sistema. Las claves del mapa corresponden a los “username” de los Usuarios (guardados en los valores de cada entrada). De esta forma, se pueden hacer consultas eficientes mediante las claves externas de las instancias.

Métodos

UsuarioCtrl():
Crea una instancia de la clase UsuarioCtrl, inicializando el mapa vacío (pues la consulta de usuarios persistidos se hará puramente “on demand”).

void crear_perfil(String username, String contrasena1, String contrasena2 , String nombre):
Crea una nueva instancia de Usuario según los parámetros de entrada y la añade al mapa de usuarios. Se verifica que se hayan puesto los parámetros correctamente y que las 2 contraseñas sean iguales.

void modificar_perfil(String username_ant, String username, String nombre):
Modifica los atributos “username” y “nombre” del usuario con clave externa “username_ant” según los parámetros de entrada. Se verifica que se hayan puesto los parámetros correctamente.

void cambiar_contrasena(String username, String contrasena, String nueva_contrasena):
Cambia el atributo “contrasena” del usuario con clave externa “username” y contraseña “contrasena” según el parámetro de entrada. Se verifica que se haya puesto el parámetro correctamente.

void borrar_perfil(String username):
Elimina la instancia de Usuario con clave externa “username”

boolean verificar_perfil(String username, String contrasena):
Comprueba que la instancia de Usuario con clave externa “username” exista y tenga el atributo contrasena con el mismo valor que el parámetro de entrada “contrasena”. Devuelve cierto si se cumple la condición, falso de lo contrario. Sirve para hacer inicios de sesión.

4.9 VectorMEV

Atributos

TermCounter:
Es un Map<String, Integer> que contiene los números de apariciones de cada palabra del texto usado en la constructora. La clave de cada entrada es la palabra en cuestión y el valor el número de repeticiones de ésta.
size:
Es un Integer que registra el tamaño del texto en número total de palabras de éste.

Métodos

VectorMEV(String texto)
Crea una instancia de la clase VectorMEV, inicializando los valores de las entradas de “TermCounter” y del entero “size” en función del parámetro de entrada. Contando cada repetición de las palabras de éste y el número total de palabras.

4.10 Corpus

Atributos

DC

Es un Map<String, Integer> que registra para cada palabra cuántos textos del Corpus la contienen. La clave de cada entrada es la palabra en cuestión y el valor el número de apariciones de ésta dentro del Corpus.

archivoVector:

Es un Map<Pair<String, String>, VectorMEV> que registra los archivos presentes en el Corpus. La clave de cada entrada es la clave externa del Archivo vectorizado (título y autor) y el valor es el VectorMEV construido a partir del contenido del Archivo vectorizado.

avgSize:

Es un Double que registra el tamaño medio en palabras de los textos del Corpus. Usado para cálculos de procesado de lenguaje natural.

Métodos

Corpus():

Crea una instancia de la clase Corpus. Inicializando todos sus atributos en función de los datos persistidos.

void anadir_archivo(String titulo, String autor, String contenido):

Añade un nuevo Archivo al Corpus. Construyendo un VectorMEV a partir del contenido entrado y asignándole el ID de Archivo en función de los parámetros de entrada. Actualiza las entradas de DC y el valor de avgSize de forma apropiada.

void borrar_archivo(String titulo, String autor):

Borra el Archivo del Corpus con clave externa “titulo” y “autor”. Actualiza las entradas de DC, archivoVector y el valor de avgSize de forma apropiada.

double comparar_archivos(String tituloA, String autorA, String tituloB, String autorB, int mode):

Compara la semejanza de los archivos A (clave externa “tituloA” y “autorA”) y B (clave externa “tituloB” y “autorB”) en función del modo de procesado de lenguaje natural (0 para TF-iDF y 1 para Okapi BM25). Devuelve un Double que representa el “coseno” de los 2 archivos según el Modelo Espacio Vectorial.

double comparar_query(String titulo, String autor, String query, int mode):

Compara la semejanza del archivo con clave externa “titulo” y “autor” y el texto entrado en el parámetro “query” en función del modo de procesado de lenguaje natural (0 para TF-iDF y 1 para Okapi BM25). Devuelve un Double que representa el “coseno” de los 2 textos según el Modelo Espacio Vectorial.

boolean existe_archivo(String titulo, String autor)

Comprueba si el Archivo con clave externa “titulo” y “autor” está registrado dentro del corpus. Devuelve cierto si es el caso y falso de lo contrario.

5. ESTRUCTURAS DE DATOS Y ALGORITMOS

En esta sección explicamos detalladamente las estructuras de datos pensadas y/o usadas para poder llevar a cabo el gestor de archivos:

5.1 Formato

Estructuras de datos

La clase Formato representa una enumeración donde se guardan los distintos tipos que puede ser un objeto archivo. Pensamos en al final implementar este concepto de esta forma porque nos facilita el acceso por parte de otras clases a los distintos formatos y la incorporación de nuevos tipos de formato.
Otra opción que planteamos fue crear dentro de la clase Archivo una clase privada con los tipos de formato pero como hemos dicho anteriormente, el acceso era más complejo de cara al uso de estos formatos por parte de otras clases.

5.2 ArchivoCtrl

Estructuras de datos

archivos(HashMap)

Para esta clase hemos implementado un HashMap<Pair<String, String>, Archivo> donde la key es un Pair de título y autor del archivo que se encuentra como valor. La razón por la cual se ha escogido esta estructura de datos ha sido porque era necesario poder crear, modificar, consultar, borrar, listar y guardar elementos complejos, en este caso usuarios. Inicialmente se contempló la idea de implementar un set de archivos pero al observar los altos costes, $\Theta(n)$, al realizar las operaciones necesarias se desechó la idea y se apostó por este map ya que sus costes eran menores, $\Theta(log(n))$.

Algoritmos

Build return list

Aquí ordenamos los elementos utilizando el sort normal asistido de un custom sort en el cual según el sort mode elegido (de 0 a 5) utiliza comparadores especiales los cuales explican cómo comparar una clase compleja que es archivo. Estos comparadores establecen si queremos ordenar por título, autor o fecha tanto de forma ascendente como descendente. Una vez tenemos los archivos ordenados hacemos un recorrido lineal del array de archivos para pasar a tener un array que contiene dos pairs (de nuestra clase pair), es decir tres elementos, título, autor y texto. Con tal de pasar una lista abstracta en la cual no se pase el archivo sino sus contenidos. A nivel temporal vemos que este hace dos recorridos lineales y una ordenación con una librería de java las cuales desarrollan un coste asintótico de $\Theta(nlog(n))$.

Listar por autor

Aquí recorremos linealmente el set que contiene todos los archivos para introducir aquellos que tengan el autor igual al proporcionado en un arraylist que posteriormente será tratado para ser devuelto según el criterio de ordenación deseado. Esta ordenación se hace en la

función build return list la cual se asegura de que el criterio de ordenación sea el correcto y además de devolver el array list en la estructura de datos deseada por el return de pair<pair<string,string>string>. El coste temporal viene determinado por un recorrido lineal y build return list, por tanto el tiempo es $\Theta(nlog(n))$.

Listar por título

Este algoritmo emplea otro arraylist auxiliar con el cual recorremos linealmente todo el set de archivos en busca de aquellos que tienen como infijo el string de título dado por el usuario. Al terminar simplemente hacemos otra llamada a build return list como en el algoritmo anterior para ordenar el resultado y entregarlo en la estructura de datos deseada. El coste temporal es igual.

Listar por semejanza

Para comenzar, obtenemos el archivo a partir del cual miraremos la semejanza. Ahora creamos un array list en el cual iremos metiendo los elementos. Con la diferencia de que crearemos un mapa ordenado de más grande a más pequeño según un double el cual representa el coseno entre los archivos. En un futuro, sería importante que el mapa tuviera como clave el archivo para evitar el raro caso de que dos archivos diferentes tengan exactamente el mismo coseno estos se pisen, todo esto manteniendo la ordenación decreciente pues es importante para el algoritmo.

Ahora recorremos todos los archivos y para cada archivo, si es diferente a nuestro archivo, lo comparamos mediante corpus, con el modo de pesos deseado. Y si el coseno es mayor que cero, es decir que estos tienen alguna relación entre ellos significativa, los introducimos dentro del mapa.

Como este mapa está ordenado de cosenos mayores a menores, los archivos del principio serán los más semejantes al archivo. Con lo cual como nos dan una n según los n archivos más relevantes que queremos ver devolveremos en el arraylist los primeros n archivos del map según el coseno.

Como se puede observar en el algoritmo, esté potencialmente puede no llegar a satisfacer la cantidad de archivos relevantes pedidos. Esto es una decisión de diseño puesto que decidimos hacer la criba de que el coseno tenga que ser mayor de 0, esto es una decisión de diseño puesto que si no hay 5 archivos relevantes no nos pareció bien mostrar 3 relevantes y otros 2 que no son suficientemente compatibles como para que tengan que ver.

Después como en todas las anteriores funciones usamos build return list para devolverlo todo.

Análisis temporal $\Theta(nlog(n) + n*r)$. (r depende del corpus)

Listar por query

Este algoritmo es análogo al anterior de listar por semejanza, con la diferencia de que en vez de comparar un archivo con todos los demás comparamos una query con todos los demás archivos. Para esto corpus ofrece una función adicional en el cual hace este tipo de

comparación y vectoriza la query para hacer sus operaciones. Todos los demás aspectos son idénticos a los anteriores.

Análisis temporal $\Theta(nlog(n) + n*r)$. (r depende del corpus)

Listar por bool

Este algoritmo emplea un arraylist para contener los archivos resultantes de la búsqueda y utiliza build return list como todos los anteriores de cara al return.

En relación a elegir qué elementos forman parte del arraylist este elemento hace servir la clase de expresiones booleanas para evaluar frase por frase de cada archivo si alguna cumple la condición expresada por la expresión. Por tanto, por cada archivo hacemos un split de su contenido en frases según unos separadores acordados. Una vez tenemos esta separación para que un archivo forme parte de la query, al menos una de sus frases ha de cumplir la condición. Con lo cual podemos ir evaluando una a una las frases con el método match de expresión booleana (el cual evalúa si la frase cumple la condición proporcionada), y si alguna de ellas lo cumple podemos añadir ese archivo al arraylist. Esta poda es altamente importante pues potencialmente reduce enormemente la cantidad de computación en ciertos casos límite.

El coste temporal es de $\mathbf{O(n*x)}$ donde n es el número de frases total entre todos los archivos y x es el número de nodos que tiene el árbol de dentro de expresión booleana, el cual es equivalente al número de operandos y variables (elementos) que hay dentro del string que genera la expresión.

5.3 UsuarioCtrl

Estructuras de datos

usuarios(HashMap)

Para esta clase hemos implementado un HashMap<String, Usuario> donde la key es el username del usuario que se encuentra como valor. La razón por la cual se ha escogido esta estructura de datos ha sido porque era necesario poder crear, modificar, consultar, borrar y guardar elementos complejos, en este caso usuarios. Inicialmente se contempló la idea de implementar un set de usuarios pero al observar los altos costes, $\Theta(n)$, al realizar las operaciones necesarias se desechó la idea y se apostó por este map ya que sus costes eran menores $\Theta(log(n))$.

5.4 VectorMEV

Estructuras de datos

TC (HashMap)

Para guardar los pesos de una forma eficiente creamos un HashMap<String, Integer> que relacione las diferentes palabras con su número de apariciones. Hay varios motivos por los que guardamos número de apariciones y no frecuencias directamente. Primeramente,

guardar enteros en vez de flotantes de doble precisión nos ahorra memoria. Finalmente, para algunos métodos como el tf-idf se usan las frecuencias de palabras, pero para otros como el Okapi BM25 se usan solo el número de apariciones.

Entonces, si mantuviéramos frecuencias, en el caso del Okapi BM25 deberíamos multiplicarlas todas por el número total de palabras en el archivo. Mantener el número de apariciones a secas nos ahorra cálculos de conversión de apariciones a frecuencias y viceversa, además de los errores de precisión que estos pueden generar.

Con esta estructura, el tiempo de consultar las apariciones de una palabra seria $\Theta(log(n))$. Mientras que el tiempo de consultar sobre todas las palabras seria $\Theta(n)$ usando un recorrido lineal con iterador.

5.5 Corpus

Estructuras de datos

DC (HashMap)

Al igual que con el HashMap TC del VectorMEV, esta estructura de datos guarda de forma eficiente el número de archivos en los que aparece cada palabra del corpus. Esto nos permite insertar y consultar elementos en tiempo $\Theta(log(n))$. Mientras que la consulta de todos los elementos sigue siendo $\Theta(n)$ usando un iterador.

archivoVector(HashMap)

Este Map<Pair<String, String>, VectorMEV> relaciona el identificador de un Archivo (String título, String autor) con el VectorMEV que guarda su contenido vectorizado. Con este Map, dado un Archivo podemos insertarlo, eliminarlo o consultarlo en el Corpus en tiempo $\Theta(log(n))$. Esto nos permite hacer la gestión eficiente del Corpus de forma dinámica de la siguiente forma:

Al añadir un Archivo, primeramente hay que crear un VectorMEV con su contenido y asignarlo al ID del Archivo. Después, sólo hace falta actualizar el Map DC para considerar las nuevas apariciones de palabras en el Corpus. Igualmente podemos hacer con borrar Archivo. Simplemente tenemos que borrar el elemento del Map archivoVector y ajustar las apariciones de DC de forma apropiada. Para modificar un Archivo podemos simplemente borrar la instancia antigua y añadir la nueva.

Para comparar 2 Archivos en Modelo de Espacio Vectorial, podemos tomar el Map DC como pivote que definirá la cardinalidad de cada vector y todos los componentes que hay que representar. A partir de aquí, podemos ir recorriendo los TC de los VectorMEV de cada Archivo e ir construyendo las componentes del vector haciendo las multiplicaciones de frecuencias en el modelo tf-idf así como los cálculos necesarios para el modelo Okapi BM25.

Estas estructuras de datos nos permiten añadir y borrar archivos en tiempo $O(r\ log(k) + log(n))$ donde r es el tamaño del texto a vectorizar en palabras, k es el tamaño del Map DC y n es el tamaño del map archivoVector. Los cálculos son de la siguiente forma: Primero hay

que actualizar las palabras de DC, para cada una se hace una consulta en ***O***(***log(k)***) y se hacen ***r*** operaciones (En el caso de añadir Archivo se incurre un coste adicional ***O(r)*** para vectorizar el texto pero asintóticamente no es relevante). Finalmente hay que insertar/consultar el elemento correspondiente en archivoVector para mantenerlo actualizado, en ***O(log(n)).***

Esta separación de apariciones de palabras en un Map TC y otro DC es esencial para poder hacer este tipo de cambios dinámicos de forma eficiente, puesto que si para cada archivo guardaramos el vector tfidf totalmente construïdo, para cada inserción o eliminación de archivos en el Corpus, cambiarían las frecuencias de todos los demás archivos. Forzando a repetir los cálculos para todos los elementos del Corpus.

Algoritmos

Idea general y estrategias de pesos:

El Modelo de Espacio Vectorial (o MEV para acortar) es bastante sencillo de entender. Se representan los textos de un conjunto (también llamado Corpus) como un vector, con el fin de poder determinar la semejanza entre 2 textos mediante el producto vectorial de éstos. Para que éste producto vectorial sea representativo, hace falta emplear estrategias de pesos que determinen la información almacenada en cada componente del vector. En este proyecto se han empleado dos estrategias: tf-idf y Okapi BM25.

tf-idf:

Ésta es una de las estrategias de procesado de lenguajes más sencillas e intuitivas que existen. Para cada palabra distinta del Corpus (es decir, cada componente del vector) los pesos se calculan de acuerdo a las siguientes fórmulas:

$$tf = \frac{num.\ apariciones\ palabra}{num.\ palabras\ en\ el\ texto}$$

$$idf = 1 + log(\frac{num.\ archivos\ en\ el\ Corpus}{num.\ archivos\ donde\ aparece\ la\ palabra})$$

$$tfidf = tf * idf$$

Como podemos ver, el *tf* a efectos prácticos registra la frecuencia de la palabra dentro del archivo mientras que *idf* registra la frecuencia de la palabra en el Corpus. De esta forma, las componentes de más peso serán aquellas que se usen de forma reiterada dentro de un archivo dado pero que no sean muy comunes dentro del conjunto total. Ésto es bastante lógico, pues palabras que no aparezcan muchas veces en un archivo no se pueden considerar representativas de éste. Sin embargo, si una palabra que aparece muchas veces en un archivo también se encuentra en los otros de forma reiterada, lo más probable es que sea algun tipo de conector como preposiciones, pronombres o artículos. Los cuales proporcionan muy poca información por naturaleza.

Okapi BM25:

Esta estrategia sigue una filosofia bastante similar a la anterior, comparando las frecuencias dentro del archivo y del Corpus. Su principal diferencia es que emplea un modelo probabilístico a la hora de calcular el componente *idf*. Es decir, en vez de usar directamente la frecuencia de una palabra en el Corpus, se apoya de ésta para determinar la probabilidad

que un determinado archivo del Corpus contenga la palabra. El cálculo de los pesos se puede representar con las siguientes formulas:

$$tf = \frac{tc*(k1+1.0)}{tc+k1*(1-b+b*\frac{D}{avgD}}$$

$$idf = ln(\frac{C-dc+0.5}{dc+0.5}+1)$$

$$BM25 = tf * idf$$

Donde *tc* representa el número de apariciones de la palabra dentro de un archivo, *dc* el número de archivos que contienen la palabra, *D* es el tamaño del archivo en palabras, *avgD* es el tamaño medio del archivo y *C* es el tamaño del Corpus en archivos.

Una de las características a destacar de ésta estrategia es que a parte de usar el modelo probabilístico para el *idf*, intenta normalizar y ajustar el *tf* al tamaño del archivo. Ésto es un punto a favor del Okapi BM25 puesto que la estrategia clásica del tf-idf es notoria por favorecer archivos de mayor longitud.

Modelo de Espacio Vectorial dinámico:

Como hemos visto, las estrategias de pesos empleadas (y muchas de las estrategias de procesamiento de lenguaje que existen) se pueden dividir en 2 componentes: El *tf* que depende mayoritariamente del *tc* y el *idf* que depende mayoritariamente del *dc*.

Inicialmente, habíamos planteado hacer una clase que se guardara el vector con las componentes del tf-idf totalmente calculadas. Sin embargo, el problema que detectamos es que cada vez que se añadiera o eliminara un archivo del Corpus se tendrian que recalcular todos los vectores, pues las frecuencias del *idf* son propensas a variar e incluso el tamaño de los vectores finales podía cambiar al añadir nuevas palabras en el Corpus.

La solución fue dividir las tareas en dos clases: un VectorMEV que es capaz de gestionar los *tc* de forma eficiente (ya que tiene el control del texto vectorizado) y un Corpus que gestiona los *dc* y se guarda todos los VectorMEV que representan archivos del Corpus vectorizados. A continuación revisaremos como se gestionan las distintas operaciones.

Añadir Archivo:

Al añadir un archivo en el Corpus, primero se coge el texto que representa su contenido y se vectoriza creando una instancia de VectorMEV. Éste dividirá el texto en una lista de palabras diferentes y el número de instancias de cada una. Luego, se consulta la lista de palabras únicas obtenida y por cada una se añade una aparición en el contador *dc* del corpus. Finalmente, se asocia el identificador del archivo con su texto vectorizado para tener rápido acceso a él.

Borrar Archivo:

Una operación bastante análoga a la anterior, se consulta la lista de palabras guardadas en el texto vectorizado del archivo a borrar. Por cada una de ellas se decrementa en 1 su contador *dc* de apariciones. Finalmente, se elimina el archivo de los datos del Corpus.

Comparar por Archivos y por Query:
A la hora de comparar la semejanza de 2 archivos o de 1 archivo con su query, primero se obtienen sus *tc* (en el caso de la query se vectorizará primero el texto para tener sus *tc*). Luego, para cada palabra del Corpus, si ésta se encuentra en el archivo, se hará el cálculo de la componente del vector de acuerdo con las fórmulas apropiadas para cada estrategia. De lo contrario, se asignará un valor de 0 al peso (pues si la *tc* es 0 la componente *tf* se vuelve 0 en ambas estrategias y por lo tanto el resultado es 0 automáticamente).

De esta forma se va construyendo el vector progresivamente. Puesto que los pesos se miden respecto a todas las palabras diferentes registradas en el Corpus, nos aseguramos que el tamaño de los 2 vectores son iguales y que ninguna palabra "local" de los archivos o la query se queda sin representación. (En caso que la query tenga palabras que no figuran en el Corpus, ésta se elimina pues no es representativa de todos modos).

Finalmente, con los vectores construïdos solo hace falta hacer el coseno mediante el producto vectorial para obtener la semejanza de los 2 elementos comparados.

El coste de esta operación seria ***O(r + d + d*log(r) + log(n))*** donde ***r*** es el número de palabras diferentes de los archivos comparados, ***d*** es el número de palabras diferentes en el corpus y ***n*** es el número de archivos en el Corpus. El primer término viene de vectorizar la query. El segundo viene de que, al hacer el cálculo del producto vectorial hay que recorrer todos los términos del vector, con tamaño ***d***. El tercer término viene de hacer ***d*** consultas de si cada archivo o query contiene una determinada palabra del corpus (cada consulta en el map tiene coste logarítmico). Finalmente, el último término es por obtener los textos vectorizados de cada archivo en el Map de archivoVector. Puesto que ***r <= d*** el coste se puede simplificar a ***~= O(d*log(r) + log(n))***.

5.6 CjtAutores

Estructuras de datos

Esta clase representa el conjunto total de todos los autores existentes, donde, para cada uno de ellos, le corresponde un conjunto de títulos de archivos del sistema. Pensamos en implementar este tipo de estructura para listar eficientemente los autores por prefijo. De esta forma, evitamos recorrer todos los *n* archivos existentes del sistema cada vez que se requiera de esta funcionalidad. El uso de esta estructura resulta más cómodo poder listar los autores y el coste sigue siendo $\Theta(n)$. Algunas de las opciones consideradas fueron:

old_autores(HashSet):
Inicialmente pensamos implementar un set de nombres de autores sin repeticiones. El coste al buscar el autor en el set para ver si existía era $\Theta(log(n))$, aspecto que nos gustó. Pero posteriormente, al pensar en la implementación de borrar un autor (que va acompañado de la eliminación de un archivo), vimos que la única forma que teníamos para ver si se tenía que eliminar del set el autor era consultando todos los autores de todos los archivos del sistema y contemplar si seguía existiendo este autor en algún otro archivo al borrado. El coste era evidentemente alto $\Theta(n)$, pues se hacía un recorrido lineal de los *n* archivos existentes.

autores(HashMap):

Como solución al caso anterior, planteamos un map de autores, cuyo segundo valor era un entero que servía de contador de apariciones del autor. De esta forma, sabemos de manera más rápida si hace falta eliminar el autor del map, ya que si el número de apariciones del autor es estrictamente mayor que 1, sabemos que aún quedan ocurrencias de ese autor en el sistema. De lo contrario, sabemos que ya no hay archivos que tengan a ese autor específico como autor y lo podemos eliminar del mapa. El coste de esto, pues, es de $\Theta(\boldsymbol{log(n)})$.

5.7 ExpBooleana

Estructuras de datos

root(Árbol Expresión (propio)):
Inicialmente para esta clase contemplamos procesar el texto de forma que mediante ciertos morfismos, pudiéramos transformar este en una expresión string que contenga unos comandos particulares que al ser procesados linealmente indiquen cómo evaluar la veracidad de una frase.

Con esta idea de procesar el string inicial y obtener una estructura de datos que nos permita evaluarla rápidamente obtuvimos árbol que ahora nos permite almacenar la expresión y poder evaluarla eficientemente.

Este árbol es un binary tree booleano el cual en sus hojas tiene las queries base que se deben de hacer sobre el texto a evaluar. Siendo estas hojas los tres tipos de elementos base: palabras sueltas, frases enteras delimitadas por comillas y conjuntos de palabras delimitados por llaves de los cuales han de estar todos presentes.

Una vez tenemos en las hojas los elementos base los nodos intermedios constituyen todas estas operaciones booleanas que se han de realizar sobre estos. Estos nodos intermedios se organizan según la precedencia natural de los operadores booleanos en java (de más prioritario a menos NOT > AND > OR). Puesto que la evaluación de la estructura de datos se hace con un DFS en el cual en el retorno de la recursión se van evaluando de las hojas a la raíz cada uno de los operandos, pondremos los operandos más prioritarios lo más cerca posible de las raíces con las que interactúan. Asimismo los paréntesis establecerán otro factor en los niveles de prioridad, los que estén rodeados de más paréntesis serán más próximos a las hojas.

Por ejemplo como puede verse en la siguiente imagen aquí tendríamos el árbol binario de expresión resultante del string “(T | F) & ! F | T | F”. Como vemos el paréntesis fuerza a dar más prioridad al or que al and. La raíz siempre es el elemento menos significativo. Esto nos servirá para crear el árbol después.

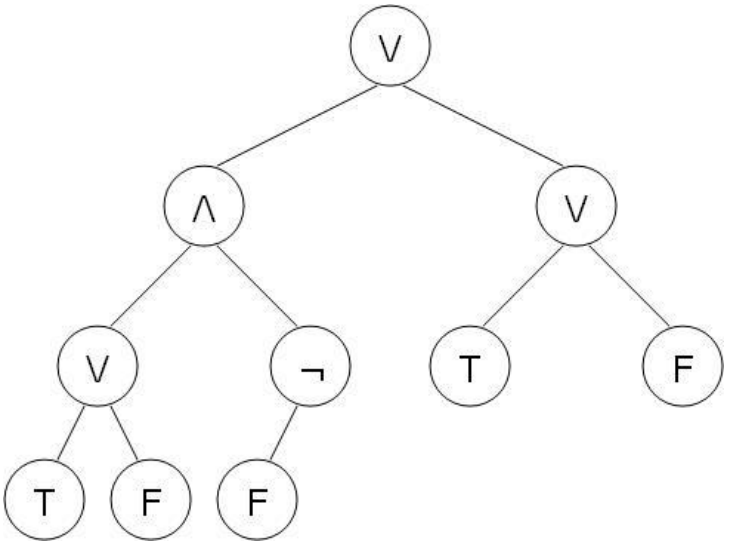


Imagen 5: Árbol binario resultante de la expresión "(T | F) & ! F | T | F"

Algoritmos

Creadora:

- **Algoritmo**

  El algoritmo de creación del árbol binario booleano crea a partir de un string el árbol expresión según las precedencias establecidas, con la posibilidad de alterarlas con la presencia de paréntesis.

  Como caso base la creadora del árbol tiene un elemento que no presenta operandos, es decir, es una hoja. Si este es el caso el elemento se trata según el tipo de hoja que sea (elemento suelto, frase, conjunto de elementos). Esto hace que el nodo hoja pueda tomar 3 tipos de valor en su interior y ser evaluado por ende de 3 formas diferentes después. Esto se hace con un indicador int el cual dependiendo de su valor indica cómo procesar el contenido de la hoja.

  El paso recursivo a realizar es el siguiente. Encontramos un splitting element (es decir un operando que nos haga partir la expresión). En primer lugar, este splitting element deberá ser el menos relevante (pues lo construimos de la raíz a la hojas los elementos de menor precedencia son los primeros que usamos para partir, para que sean los últimos en realizarse al evaluar. Esto lo hacemos con una función find split que encuentra el elemento rodeado de la menor cantidad de paréntesis y menos importante en precedencia. Una vez tenemos este elemento lo dejamos como contenido del nodo actual y recursivamente cogemos las mitades restantes y las mandamos a los nodos hijos de este para que estos vayan partiendo sucesivamente hasta no tener más.

  En todo este proceso además vamos eliminando espacios innecesarios, y comprobando que tanto la expresión como la parentización sean correctas. Si no es así se lanza una excepción que después será recogida en clases superiores.

- **Tiempo**
  A nivel asintótico, la recurrencia hace una serie de tareas lineales y después pasa en caso recursivo dos mitades. El problema radica en que estas dos mitades no necesariamente son iguales en tamaño, cosa que nos aseguraría un **O(*nlogn*).** Como no podemos asegurar eso en caso peor llegamos a la cota del **O(*n*^*2*)** pues realmente la recurrencia potencialmente podría llegar a ser **T(*n*) = T(*n-1*) + O(*n*)** si los splitting elements son especialmente malos.

Match:

- **Algoritmo**
  Como vimos anteriormente este es básicamente un DFS que devuelve booleanos.
  El caso base de nuevo es llegar a las hojas, una vez llegamos a estas hacemos la consulta contains de lo que sea que se nos pida y devolvemos un valor booleano a los nodos superiores para que estos operen sobre ellos.
  Para los nodos intermedios este antes de hacer el salto recursivo dejan indicado un return con la operación booleana a realizar (not, and o or) y deja la responsabilidad de evaluar el resto del subárbol a cada una de sus mitades. Así cuando la traca recursiva vuelva de las hojas arriba con sus respectivos valores booleanos este nodo operará sobre ellos y pasará el testigo a sus superiores hasta llegar a la raíz.

- **Tiempo**
  El tiempo del algoritmo es O($V + E$) siendo V el número de vértices y E el número de aristas. Como este es un árbol sabemos que su número de aristas es exactamente |V| - 1 lo cual hace que el tiempo sea O($V + V - 1$) = O(V), es decir lineal segun el número de aristas presentes en el árbol.

- **Idea general:**
  Por tanto esta clase lo que hace es crear una estructura de datos en tiempo cuadrático en relación al tamaño de string de entrada como peor caso, con la posibilidad de hacer pequeñas podas y optimizaciones en versiones futuras que acerquen este tiempo a uno más linearítmico (*n(log n)*).
  Para que luego esta expresión pueda ser evaluada en tiempo lineal de forma inequívoca.

5.8 ExpBooleanaCtrl

Estructuras de datos

expresiones(HashMap):

Puesto que era necesario poder crear, borrar y guardar elementos complejos (objetos del tipo ExpBooleana) necesitamos encontrar una estructura que gestione este conjunto. Como este elemento es complejo y no hay una forma evidente de compararlo ni ordenarlo dentro de un conjunto para operar sobre él de una determinada manera, usamos el string que lo identifica como clave y este como contenido de un map. Este string lo identifica correctamente puesto que es la base con la que se crea la estructura de ExpBooleana, y los algoritmos que lo crean son deterministas. Dos expresiones booleanas con el mismo string de base son iguales.

6. TESTS

Descripción de juegos de pruebas

6.1 Archivo

TestArchivoYConsulta()
Objeto de la prueba: Test de la constructora y las consultoras de los atributos del objeto creado Archivo.
Valores estudiados: Se comprueba que se cree correctamente el archivo con todos sus respectivos atributos. Se comprueba que se realicen correctamente las consultas de cada uno de dichos atributos.
Operativa: Se crea una instancia de Archivo cuyos atributos corresponden con los valores introducidos. Se comprueba que los atributos consultados sean correctos.

TestModificacionArchivo()
Objeto de la prueba: Test de la modificación de un archivo
Valores estudiados: Se comprueba que los atributos del archivo se actualicen correctamente con los valores introducidos como parámetros diferentes a los valores del atributo correspondiente.
Operativa: Se crea un archivo y se introducen nuevos valores para modificar este. Se comprueba que los atributos estén actualizados y correspondan a los nuevos valores introducidos para la modificación.

6.2 ArchivoCtrl

Para hacer este test, se ha realizado antes el test de la clase Archivo, Corpus y VectorMEV.

TestAltaArchivo()
Objeto de la prueba: Test del método alta_archivo de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que se crea el archivo con los parámetros introducidos y que se añade correctamente al set y al corpus.
Operativa: Se crea un usuario y se verifica que existe en el set y en el corpus, es decir, que se ha creado y guardado correctamente.

TestModificarArchivoSinCambiarID()
Objeto de la prueba: Test del método modificar_archivo de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que se modifican los parámetros no identificativos del archivo por los introducidos tanto en el set como en el corpus.
Operativa: Se crea un archivo con unos valores y se añade al set y al corpus. Se modifican valores no identificativos por otros y se comprueba que la modificación de los valores se haya realizado correctamente.

TestModificarArchivoCambioID()
Objeto de la prueba: Test del método modificar_archivo de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que se modifican todos los parámetros del archivo por los introducidos tanto en el set como en el corpus.
Operativa: Se crea un archivo con unos valores y se añade al set y al corpus. Se modifican todos valores por otros y se comprueba que la modificación de los valores se haya realizado correctamente.

TestBorrarArchivo()
Objeto de la prueba: Test del método borrar_archivo de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que se borra un archivo y deja de estar en el set y en el corpus.
Operativa: Se crea un archivo con unos valores y se añade al set y al corpus. Se borra este mismo usuario y se verifica que ya no forma parte del set ni del corpus.

TestConsultaFormatoArchivo()
Objeto de la prueba: Test del método consulta_formato_archivo de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve el formato del archivo solicitado.
Operativa: Se crea un archivo con unos valores y se comprueba que el formato que devuelve sea el mismo que el introducido al crear el archivo.

TestConsultaContenidoArchivo()
Objeto de la prueba: Test del método consulta_contenido_archivo de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve el contenido del archivo solicitado.
Operativa: Se crea un archivo con unos valores y se comprueba que el contenido que devuelve sea el mismo que el introducido al crear el archivo.

TestListarPorAutor()
Objeto de la prueba: Test del método listar_por_autor de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos cuyo autor es el introducido como parámetro.
Operativa: Se crean 3 archivos, dos con el autor introducido como parámetro y otro con otro autor, y se comprueba que la salida de la lista con los datos de los archivos correspondan a los que tienen el autor introducido como parámetro.

TestListarPorTitulo()
Objeto de la prueba: Test del método listar_por_titulo de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos cuyo título contiene el parámetro introducido.
Operativa: Se crean 3 archivos, dos con el parámetro introducido como parte del título y otro con otro título, y se comprueba que la salida de la lista con los datos de los archivos correspondan a los que tienen el parámetro introducido como parte del título.

TestListarPorSemejanzaWeightMode0()
Objeto de la prueba: Test del método listar_por_semejanza de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los k archivos más parecidos al introducido en los parámetros con la estrategia de pesos en modo 0.

Operativa: Se crean 3 archivos, dos con el contenido similar y otro con el contenido totalmente diferente, y se comprueba que la salida de la lista con los datos de los archivos corresponda al archivo con contenido similar al introducido como parámetro.

TestListarPorSemejanzaWeightMode1()

Objeto de la prueba: Test del método listar_por_semejanza de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los k archivos más parecidos al introducido en los parámetros con la estrategia de pesos en modo 1.
Operativa: Se crean 3 archivos, dos con el contenido similar y otro con el contenido totalmente diferente, y se comprueba que la salida de la lista con los datos de los archivos corresponda al archivo con contenido similar al introducido como parámetro.

TestListarPorQuery()

Objeto de la prueba: Test del método listar_por_query de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos que contienen la query introducida como parámetro.
Operativa: Se crean 3 archivos, dos con contenido de la query y otro con el contenido totalmente diferente, y se comprueba que la salida de la lista con los datos de los archivos corresponda a los archivos con contenido similar a la query introducida como parámetro.

TestListarPorBool()

Objeto de la prueba: Test del método listar_por_bool de la clase ArchivoCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos cuyo contenido cumple con la expresión booleana introducida como parámetro.
Operativa: Se crean 3 archivos, dos que cumplen con la expresión booleana y otro que no, y se comprueba que la salida de la lista con los datos de los archivos corresponda a los archivos cuyo contenido cumple con la expresión booleana introducida como parámetro.

6.3 Usuario

TestCrearUsuarioYConsulta()

Objeto de la prueba: Test de la creadora y de los métodos consulta_username, consulta_contrasena y consulta_nombre de la clase Usuario .
Valores estudiados: Se comprueba que se crea el usuario con los parámetros introducidos.
Operativa: Se crea un usuario con los valores de los parámetros y se comprueba que se haya creado correctamente.

TestModificarUsuario()

Objeto de la prueba: Test del método modificar_usuario de la clase Usuario.
Valores estudiados: Se comprueba que se modifique el usuario con los parámetros introducidos.
Operativa: Se crea un usuario con unos valores, se modifican estos valores por otros y se comprueba que la modificación de los valores se haya realizado correctamente.

TestModificarUsuarioNull()
Objeto de la prueba: Test del método modificar_usuario de la clase Usuario.
Valores estudiados: Se comprueba que no se modifique el usuario con los parámetros null introducidos.
Operativa: Se crea un usuario con unos valores, se intenta modificar estos valores por otros con valor null y se comprueba que la modificación de los valores no se haya realizado.

TestCambiarContrasena()
Objeto de la prueba: Test del método cambiar_contrasena de la clase Usuario.
Valores estudiados: Se comprueba que se modifique la contraseña del usuario con el parámetro introducido.
Operativa: Se crea un usuario con unos valores, se modifica la contraseña y se comprueba que la modificación se haya realizado correctamente.

TestCambiarContrasenaNull()
Objeto de la prueba: Test del método cambiar_contraseña de la clase Usuario.
Valores estudiados: Se comprueba que no se modifique la contraseña del usuario con el parámetro introducido.
Operativa: Se crea un usuario con unos valores, se intenta modificar la contraseña por una con valor null y se comprueba que la modificación no se haya realizado.

6.4 UsuarioCtrl

Para hacer este test, se ha realizado antes el test de la clase Usuario.

TestCrearYExistePerfil()
Objeto de la prueba: Test de los métodos crear_perfil y existe_perfil de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que se crea el usuario con los parámetros introducidos y que se añade correctamente al set.
Operativa: Se crea un usuario y se verifica que existe en el set, es decir, que se ha creado y guardado correctamente.

TestCrearPerfilErrorUsername()
Objeto de la prueba: Test del método crear_perfil de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que no se crea el usuario con un username inválido, que no se añade al set y que se lanza la excepción correspondiente.
Operativa: Se intentan crear dos usuarios, el primero se crea correctamente, mientras que el segundo al tener un username que ya está en uso no se crea y se verifica que este segundo no existe en el set y que se lanza la excepción correspondiente.

TestCrearPerfilErrorContrasena()
Objeto de la prueba: Test del método crear_perfil() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que no se crea el usuario con contraseña inválida, que no se añade al set y que se lanza la excepción correspondiente.

Operativa: Se intenta crear un usuario con contraseñas diferentes y se verifica que no existe en el set y que se lanza la excepción correspondiente.

TestModificarPerfil()

Objeto de la prueba: Test del método modificar_perfil() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que se modifica el usuario con los parámetros introducidos.
Operativa: Se crea un usuario con unos valores y se añade al set. Se modifican estos valores por otros y se comprueba que la modificación de los valores se haya realizado correctamente.

TestModificarPerfilErrorUsername()

Objeto de la prueba: Test del método modificar_perfil() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que no se modifica el usuario cuando el parámetro username introducido es inválido y que se lanza la excepción correspondiente.
Operativa: Se crean dos usuarios con unos valores y se añaden al set. Se intenta modificar los valores del segundo pero al tener un username que ya está en uso no se modifica y se lanza la excepción correspondiente.

TestCambiarContrasena()

Objeto de la prueba: Test del método cambiar_contrasena() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que se modifica la contraseña del usuario con el parámetro introducido.
Operativa: Se crea un usuario con unos valores y se añade al set. Se modifica la contraseña y se comprueba que la modificación se haya realizado correctamente.

TestCambiarContrasenaError()

Objeto de la prueba: Test del método cambiar_contrasena() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que no se modifica la contraseña del usuario cuando los parámetros contraseñas introducidos son inválidos y que se lanza la excepción correspondiente.
Operativa: Se crea un usuario con unos valores y se añade al set. Se intenta modificar la contraseña, pero si la contraseña actual es incorrecta no se modifica y se lanza la excepción correspondiente.

TestBorrarPerfil()

Objeto de la prueba: Test del método borrar_perfil() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que se borra un usuario y deja de estar en el set
Operativa: Se crea un usuario con unos valores y se añade al set. Se borra este mismo usuario y se verifica que ya no forma parte del set.

TestVerificaPerfilExiste()

Objeto de la prueba: Test del método verificar_perfil() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que devuelve true al insertar como parámetros un usuario y contraseña que existe.
Operativa: Se crea un usuario y se realiza la comprobación.

TestVerificaPerfilNoExiste()

Objeto de la prueba: Test del método verificar_perfil() de la clase UsuarioCtrl.
Valores estudiados: Se comprueba que lanza la excepción al intentar verificar un perfil con parámetros incorrectos.
Operativa: Se realiza la comprobación sin haber creado ningún usuario y se lanza la excepción correspondiente.

6.5 VectorMEV

TestCreacionYConsulta()

Objeto de la prueba: Test de la constructora y de los “getters” de VectorMEV. Con tal de verificar que los textos se vectorizan correctamente.
Valores estudiados: Se comprueba que el texto se vectoriza correctamente. Demostrando que cuenta adecuadamente el número de apariciones de cada palabra y el número de palabras totales en el texto de entrada.
Operativa: Se define un texto de ejemplo y se llama a la constructora de VectorMEV para que lo vectorize. Acto seguido se comprueba que todas las apariciones de palabra se hayan registrado correctamente y, finalmente, que el número de palabras del archivo sea el apropiado.

6.6 Corpus

Para hacer este test, se ha realizado antes el test de la clase VectorMEV.

TestAnadirArchivos()

Objeto de la prueba: Test de la constructora y del método anadir_archivo(). Con tal de verificar que se añaden correctamente en el Corpus
Valores estudiados: Se comprueba que el Corpus registra correctamente la inserción de nuevos Archivos
Operativa: Se crea un nuevo Corpus con su constructora por defecto. Acto seguido se verifica que no reconoce ninguno de los archivos que se quieren añadir. Finalmente, se llama a los métodos para añadir los archivos y se verifica que ahora sí son reconocidos por el Corpus.

TestArchivoYaExisteException()

Objeto de la prueba: Test de la excepción ArchivoYaExisteException como caso erróneo de añadir archivos en el Corpus.
Valores estudiados: Se comprueba que el Corpus lanza correctamente la excepción ArchivoYaExisteException cuando se añade un archivo con ID repetido.
Operativa: Se crea un nuevo Corpus con su constructora por defecto. Acto seguido se verifica que no reconoce ninguno de los archivos que se quieren añadir. Luego, se llama a los métodos para añadir los archivos y se verifica que ahora sí son reconocidos por el Corpus. Finalmente se intenta añadir un archivo con ID repetido (y contenido arbitrario) y se comprueba que se lanza la excepción correctamente

TestAnadirYBorrarArchivos()

***Objeto de la prueba:** Test del método anadir_archivo() y borrar_archivo(). Con tal de verificar que los archivos se gestionan correctamente en el Corpus y de forma dinámica.
***Valores estudiados:** Se comprueba que el Corpus registra correctamente la inserción y eliminación dinámica de archivos.
***Operativa:** Se crea un nuevo Corpus con su constructora por defecto. Acto seguido se verifica que no reconoce ninguno de los archivos que se quieren añadir. Luego, se llama a los métodos para añadir los archivos y se verifica que ahora sí son reconocidos por el Corpus. Finalmente se borran ciertos archivos de éste y se comprueba que ahora ya no son reconocidos.

TestCompararArchivosBothStrategies()

***Objeto de la prueba:** Test del método comparar_archivos() con ambas estrategias de asignación de pesos. Con tal de verificar que el Corpus gestiona los pesos de los archivos y calcula el coseno resultante del producto vectorial de forma apropiada.
***Valores estudiados:** Se comprueba que el Corpus devuelve un valor correcto de coseno al comparar dos archivos con ambas estrategias de asignación de pesos. Probando casos ordinarios así como casos extremos como que dos archivos no tengan nada que ver o que sean el mismo (cosenos de 0 y 1 respectivamente).
***Operativa:** Se crea un nuevo Corpus con su constructora por defecto. Acto seguido se añaden 4 archivos de la siguiente forma: 1 y 2 tienen contenidos relacionados entre sí. De la misma forma, 3 y 4 también están relacionados entre sí, pero entre la primera pareja de archivos y la segunda no hay ningun tipo de relación. Una vez finalizadas las inserciones, se verifican los casos ordinarios comparando los archivos de una misma pareja entre sí (los cuales daran un coseno entre 0 y 1). Finalmente, se prueban los casos extremos comparando un archivo con él mismo (coseno de 1) y comparando un archivo de la primera pareja con uno de la segunda (coseno de 0).

TestCompararQueryBothStrategies()

Objeto de la prueba: Test del método comparar_query() con ambas estrategias de asignación de pesos. Con tal de verificar que el Corpus gestiona los pesos del archivo y la query de forma apropiada, además de calcular el coseno resultante del producto vectorial correctamente.
Valores estudiados: Se comprueba que el Corpus devuelve un valor correcto de coseno al comparar un archivo con una query en ambas estrategias. Probando casos ordinarios así como casos extremos como que la query no tenga nada que ver con el archivo o, por lo contrario, que tenga el mismo texto (cosenos de 0 y 1 respectivamente)
Operativa: Se crea un nuevo Corpus con su constructora por defecto. Acto seguido se añaden 4 archivos de la siguiente forma: 1 y 2 tienen contenidos relacionados entre sí. De la misma forma, 3 y 4 también están relacionados entre sí, pero entre la primera pareja de archivos y la segunda no hay ningún tipo de relación. Una vez finalizadas las inserciones, se crea una query que está relacionada con la primera pareja de archivos pero no con la segunda. Finalmente, se comprueban los casos ordinarios comparando la query con archivos de la primera pareja y casos extremos comprobando la query con archivos de la segunda pareja (coseno de 0) y cogiendo el contenido del archivo a comparar como query (coseno de 1).

TestArchivoNoExisteException()

Objeto de la prueba: Test de la excepción ArchivoNoExisteException como caso erróneo de intentar usar archivos no registrados en el Corpus.
Valores estudiados: Se comprueba que el Corpus lanza correctamente la excepción ArchivoNoExisteException cuando se opera sobre un archivo con ID no registrada en el Corpus.
Operativa: Se crea un nuevo Corpus con su constructora por defecto. Acto seguido se llama a la operación borrar_archivo() con un archivo de ID arbitraria. Ya que el Corpus está vacío, al intentar acceder a cualquier archivo (para realizar su borrado) se lanzará la excepción ArchivoNoExisteException. Este test sirve también para verificar la excepción en comparar_archivos() y comparar_query() ya que ambos tienen como subcaso el "getter" del archivo que es el que lanza la excepción.

6.7 CjtAutores

TestInsertarYListarAutorSinPrefijo()

Objeto de la prueba: Test de la constructora, del método insertar_autor() de un autor y listar_autores() de todos los autores del sistema.
Valores estudiados: Se comprueba que se cree correctamente el listado con todos los autores y títulos insertados, sin repeticiones de nombres.
Operativa: Se crea un listado de autores con los autores que han sido insertados y los títulos de archivo que corresponden a cada uno. Al tener un prefijo vacío el listado contiene todos los autores existentes.

TestInsertarAutorConTituloExistente()

Objeto de la prueba: Test del método insertar_autor() de autor ya existente (Tratado de Excepción)
Valores estudiados: Se comprueba que al insertar una pareja de autor,título ya existentes, salte la excepción de que el autor con ese título ya existe en el sistema.
Operativa: Se crea una pareja de valores autor y título, representando el id de un archivo. Se insertan en el conjunto de autores y se vuelve a insertar para comprobar que salte la excepción.

TestModificarTituloDeAutorNoExistente()

Objeto de la prueba: Test del método modificar_titulo_autor() de un autor que no existe en el conjunto (Tratado de excepción)
Valores estudiados: Se comprueba que al intentar modificar el título de un autor que no existe en el conjunto de autores, se ejecute la excepción de que el autor no existe en el sistema.
Operativa: Se intenta modificar un autor que no está presente en el conjunto de autores y se comprueba que salte la excepción.

TestModificarTituloNoExistenteDeAutor()

Objeto de la prueba: Test del método modificar_titulo_autor() de un título no existente en el conjunto de títulos de un autor.
Valores estudiados: Se comprueba que al intentar modificar un título que no pertenece al conjunto de títulos del autor correspondiente, salte la excepción de que el título no corresponde al autor indicado.

Operativa: Se crea un listado de autores al insertar un autor junto con un título. Se intenta modificar un título que no forma parte del conjunto de títulos del autor y se comprueba que salte la excepción.

TestModificarTituloDeAutorConUnoExistente()

Objeto de la prueba: Test del método modificar_titulo_autor() de un título de un autor con otro título ya existente.
Valores estudiados: Se comprueba que al intentar modificar el título de un autor, si el nuevo título corresponde a otro existente en el conjunto de títulos del autor, salte la excepción de que el archivo ya existe.
Operativa: Se crea un listado de autores al insertar un autor junto con varios títulos. Se intenta modificar un título del autor con otro nombre que ya forma parte del conjunto y se comprueba que salte la excepción.

TestModificarTituloAutor()

Objeto de la prueba: Test del método modificar_titulo_autor().
Valores estudiados: Se comprueba que se modifica correctamente el título del autor indicado.
Operativa: Se crea un listado de autores al insertar un autor junto con un título. Se comprueba que del autor indicado, se sustituye el título escogido por uno nuevo.

TestEliminarAutorCompletamente()

***Objeto de la prueba:** Test del método eliminar_autor().
***Valores estudiados:** Se comprueba la correcta eliminación del autor entrado siendo su frecuencia 1 (se ha insertado el autor 1 vez en total).
***Operativa:** Se crea un listado de autores al insertar el autor. Al ser insertado solo una vez, al realizar la eliminación, se comprueba que el listado quede vacío.

TestEliminarAutor()

Objeto de la prueba: Test del método eliminar_autor().
Valores estudiados: Se comprueba la correcta eliminación del autor entrado siendo su frecuencia mayor que 1 (se ha insertado el autor más de una vez en total, es decir, hay más de un archivo con el mismo autor pero distinto título).
Operativa: Se crea un listado de autores al insertar el autor. Al ser insertado 2 veces, al realizar una eliminación, se comprueba que el listado contenga el autor.

TestEliminarAutorNoExistente()

Objeto de la prueba: Test del método eliminar_autor() de autor no existente en el conjunto de autores
Valores estudiados: Se comprueba que al intentar eliminar un autor que no forma parte del conjunto de autores salte la excepción de que el autor no existe.
Operativa: Se intenta eliminar un autor no existente en el sistema y se comprueba que salte la excepción.

TestEliminarAutorDeTituloNoExistente()

Objeto de la prueba: Test del método eliminar_autor() de un autor de un título no existente
Valores estudiados: Se comprueba que al intentar eliminar un título no existente del autor, salte la excepción de que el título no corresponde al autor.

Operativa: Se crea un listado de autores al insertar un autor junto con un título. Se intenta borrar un título que no forma parte del conjunto de títulos del autor y se comprueba que salte la excepción.

TestListarAutorConPrefijo()

Objeto de la prueba: Test del método listar_autores() por un prefijo entrado.
Valores estudiados: Se comprueba que se listen correctamente los autores cuyo prefijo cumple con el indicado.
Operativa: Se crea un listado de autores con los autores insertados. Se comprueba que se listen únicamente los autores que contengan el mismo prefijo que el entrado.

TestListarTitulosAutor()

Objeto de la prueba: Test del método titulos_autor() de un autor entrado.
Valores estudiados: Se comprueba que se listen correctamente los títulos del autor indicado.
Operativa: Se crea un listado de autores con el autor insertado junto con varios títulos. Se comprueba que se listen todos los títulos existentes del autor indicado.

TestListarTitulosAutorNoExistente()

Objeto de la prueba: Test del método titulos_autor() de un autor no existente (Tratado de excepción).
Valores estudiados: Se comprueba que al intentar listar los títulos de un autor inexistente, salte la excepción de que no existe el autor.
Operativa: Se crea un listado de autores con el autor insertado junto con varios títulos. Se al intentar listar los títulos de un autor que no forma parte del conjunto de autores, salte la excepción.

6.8 ExpBooleana

testWithoutParenthesis()

Objeto de la prueba: Test de la constructora y el método to_String para comprobar su correcta creación. El propósito principal de la prueba consiste en comprobar que la evaluación de match es correcta bajo el supuesto de que la expresión es compleja pero sin llegar a cambiar su precedencia natural
Valores estudiados: Se comprueba que las funciones creadora, to_String y match funcionen correctamente.
Operativa: Se crea una nueva instancia de ExpBooleana con la expresión a evaluar, en este caso una expresión sin paréntesis para evaluar la precedencia natural. A continuación comprobamos que la creación sea adecuada y evaluamos con match dos frases, una cierta y otra falsa y comprobamos que efectivamente la expresión evalúa el texto de forma satisfactoria. De ahora en adelante todos los tests de ExpBooleana siguen este formato con una expresión y frases ciertas y falsas particulares.

testWithParenthesis()

Objeto de la prueba: Igual a los anteriores pero esta vez la sentencia de match ha de ser correcta bajo el supuesto de que la expresión es compleja y presenta paréntesis que alteran la precedencia natural de la expresión.

Valores estudiados: Se comprueba que las funciones creadora, to_String y match funcionen correctamente.
Operativa: Igual que antes pero en este caso evaluamos una expresión con paréntesis que alteran la precedencia natural de los operandos.

testTooManyParenthesis()

Objeto de la prueba: Igual a los anteriores pero esta vez la sentencia de match ha de ser correcta bajo el supuesto de que la expresión es compleja y presenta paréntesis redundantes (pero correctos). Es decir, elementos como este "((hola))" donde hay paréntesis que no aportan nada pero han de ser tratados.
Valores estudiados: Se comprueba que las funciones creadora, to_String y match funcionen correctamente.
Operativa: Igual que antes pero en este caso evaluamos una expresión en que algunos paréntesis son redundantes.

testComillas()

Objeto de la prueba: Igual a los anteriores pero esta vez la sentencia de match ha de ser correcta bajo el supuesto de que la expresión es compleja y utiliza el elemento de las comillas con chars que podrían ser evaluados como operandos pero no deben serlo pues forman parte de la búsqueda.
Valores estudiados: Se comprueba que las funciones creadora, to_String y match funcionen correctamente.
Operativa: Igual que antes pero en este caso evaluamos una expresión con el elemento de las comillas que contiene elementos que podrían considerarse operandos.

testBrackets()

Objeto de la prueba: Igual a los anteriores pero esta vez la sentencia de match ha de ser correcta bajo el supuesto de que la expresión es compleja y utiliza los brackets como elemento de búsqueda donde todo lo que hay dentro ha de estar presente en la sentencia.
Valores estudiados: Se comprueba que las funciones creadora, to_String y match funcionen correctamente.
Operativa: Igual que antes pero en este caso evaluamos una expresión donde se utiliza el elemento de los brackets para comprobar que todas las palabras de ese conjunto están presentes.

testDumbSpacing()

Objeto de la prueba: Igual a los anteriores pero esta vez la sentencia de match ha de ser correcta bajo el supuesto de que la expresión es compleja como todas las anteriores y contiene largas cadenas de espacios que no aportan nada.
Valores estudiados: Se comprueba que las funciones creadora, to_String y match funcionen correctamente.
Operativa: Igual que antes pero en este caso evaluamos una expresión donde hay una gran cantidad de espacios que no aportan nada.

testsExcepciones()

Objeto de la prueba: En este caso lo que comprobamos son los distintos tipos de excepciones especiales que implementa la clase con tal de asegurar su correcto funcionamiento.

Valores estudiados: Se comprueban los posibles errores que desencadena la creación.
Operativa: Una creadora por cada posible excepción con fallos diversos. En primer lugar una con un carácter ilegal (separador de frases). Luego, una con un mal uso de operandos (operadores binarios con un solo argumento etc). A continuación, una con mal uso de paréntesis. Y para acabar una con mal uso de comillas (abiertas pero nunca cerradas).

6.9 ExpBooleanaCtrl

Para hacer este test, se ha realizado antes el test de la clase ExpBooleana.

testInsertar()
Objeto de la prueba: Test de la constructora para ver que se realiza de forma correcta, método de alta para ver la introducción y listado para comprobar si el estado del conjunto es correcto.
Valores estudiados: Se comprueban la constructora, el método de alta y el listado en set.
Operativa: Una creadora de ExpBooleanaCtrl y otra de Set para comprobar el estado del conjunto. Vamos añadiendo elementos en ambos con sus respectivas funciones y luego los comparamos.

testInsertar()
Objeto de la prueba: Test de la constructora para ver que se realiza de forma correcta, método de alta para ver la introducción y baja para ver la eliminación y listado para comprobar si el estado del conjunto es correcto.
Valores estudiados: Se comprueban la constructora, el método de alta, baja y el listado en set.
Operativa: Una creadora de ExpBooleanaCtrl y otra de Set para comprobar el estado del conjunto. Vamos añadiendo y quitando elementos en ambos con sus respectivas funciones y luego los comparamos.

testsExtraer()
Objeto de la prueba: Test de la constructora para ver que se realiza de forma correcta, método de alta para ver la introducción y el método de consulta para extraer el ExpBooleana.
Valores estudiados: Se comprueban la constructora, el método de alta y extracción de elementos.
Operativa: Una creadora de ExpBooleanaCtrl donde introducimos un elemento, a posteriori creamos una ExpBooleana para comprobar que cuando la extraemos de ExpBooleanaCtrl es la misma.

6.10 DomainCtrl

Antes de realizar el test de DomainCtrl, se han realizado todos los demás tests.

CrearYExistePerfilTest()
Objeto de la prueba: Test de los métodos crear_perfil() y existe_perfil() de la clase DomainCtrl.

Valores estudiados: Se comprueba que se crea el usuario con los parámetros introducidos.
Operativa: Se crea un usuario y se verifica que existe, es decir, que se ha creado y guardado correctamente.

ModificarPerfilTest()
Objeto de la prueba: Test del método modificar_perfil() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se modifica el usuario con los parámetros introducidos.
Operativa: Se crea un usuario con los valores introducidos. Se modifican estos valores por otros y se comprueba que la modificación de los valores se haya realizado correctamente.

CambiarContrasenaTest()
Objeto de la prueba: Test del método cambiar_contrasena() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se modifica la contraseña del usuario con el parámetro introducido.
Operativa: Se crea un usuario con los valores introducidos. Se modifica la contraseña y se comprueba que la modificación de los valores se haya realizado correctamente.

BorrarPerfilTest()
Objeto de la prueba: Test del método borrar_perfil() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se borra un usuario.
Operativa: Se crea un usuario con unos valores y se borra este mismo usuario y se verifica que ya no existe.

VerificaPerfilTest()
Objeto de la prueba: Test del método verificar_perfil() de la clase DomainCtrl.
Valores estudiados: Se comprueba que devuelve true al insertar como parámetros un usuario y contraseña que existe.
Operativa: Se crea un usuario y se realiza la comprobación.

AltaArchivoYConsultasTest()
Objeto de la prueba: Test del método alta_archivo() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se crea el archivo con los parámetros introducidos y que se añade correctamente al archivoCtrl y al corpus.
Operativa: Se crea un usuario y se verifica que existe en el archivoCtrl y en el corpus, es decir, que se ha creado y guardado correctamente.

ModificarArchivoSinCambiarIDTest()
Objeto de la prueba: Test del método modificar_archivo() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se modifican los parámetros no identificativos del archivo por los introducidos tanto en el set como en el corpus.
Operativa: Se crea un archivo con unos valores y se añade al archivoCtrl y al corpus. Se modifican valores no identificativos por otros y se comprueba que la modificación de los valores se haya realizado correctamente.

ModificarArchivoCambioIDTest()
Objeto de la prueba: Test del método modificar_archivo() de la clase DomainCtrl.

Valores estudiados: Se comprueba que se modifican todos los parámetros del archivo por los introducidos tanto en el archivoCtrl como en el corpus.
Operativa: Se crea un archivo con unos valores y se añade al set y al corpus. Se modifican todos valores por otros y se comprueba que la modificación de los valores se haya realizado correctamente.

BorrarArchivoTest()
Objeto de la prueba: Test del método borrar_archivo() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se borra un archivo y deja de estar en el archivoCtrl y en el corpus.
Operativa: Se crea un archivo con unos valores y se añade al set y al corpus. Se borra este mismo usuario y se verifica que ya no forma parte del archivoCtrl ni del corpus.

ListarAutoresTest()
Objeto de la prueba: Test del método listar_autores() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se listan todos los autores de CjtAutores.
Operativa: Se crea un archivo con unos valores y se listan los autores. Se comprueba que en la lista se encuentra el autor del archivo creado.

ListarPorAutorTest()
Objeto de la prueba: Test del método listar_por_autor() de la clase DomainCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos cuyo autor es el introducido como parámetro.
Operativa: Se crean 3 archivos, dos con el autor introducido como parámetro y otro con otro autor, y se comprueba que la salida de la lista con los datos de los archivos correspondan a los que tienen el autor introducido como parámetro.

ListarPorTituloTest()
Objeto de la prueba: Test del método listar_por_titulo() de la clase DomainCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos cuyo título contiene el parámetro introducido.
Operativa: Se crean 3 archivos, dos con el parámetro introducido como parte del título y otro con otro título, y se comprueba que la salida de la lista con los datos de los archivos correspondan a los que tienen el parámetro introducido como parte del título.

ListarPorSemejanzaTest()
Objeto de la prueba: Test del método listar_por_semejanza() de la clase DomainCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los k archivos más parecidos al introducido en los parámetros.
Operativa: Se crean 3 archivos, dos con el contenido similar y otro con el contenido totalmente diferente, y se comprueba que la salida de la lista con los datos de los archivos corresponda al archivo con contenido similar al introducido como parámetro.

ListarPorQueryTest()
Objeto de la prueba: Test del método listar_por_query() de la clase DomainCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos que contienen la query introducida como parámetro.

Operativa: Se crean 3 archivos, dos con contenido de la query y otro con el contenido totalmente diferente, y se comprueba que la salida de la lista con los datos de los archivos corresponda a los archivos con contenido similar a la query introducida como parámetro.

ListarPorBoolTest()

Objeto de la prueba: Test del método listar_por_bool() de la clase DomainCtrl.
Valores estudiados: Se comprueba que devuelve una lista con título, autor y última fecha de modificación de los archivos cuyo contenido cumple con la expresión booleana introducida como parámetro.
Operativa: Se crean 3 archivos, dos que cumplen con la expresión booleana y otro que no, y se comprueba que la salida de la lista con los datos de los archivos corresponda a los archivos cuyo contenido cumple con la expresión booleana introducida como parámetro.

AltaExpresionYListarExpresionesTest()

Objeto de la prueba: Test del método alta_expresion() y listar_expresiones() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se crea la expresión booleana introducida y que se añade correctamente a ExpBooleanaCtrl mediante el listar_autores.
Operativa: Se crean dos expresiones con los valores de los parámetros y se crea un set con estas. Se hace el listar_autores y se comprueba que el resultado sea igual al set creado.

ModificarExpresionTest()

Objeto de la prueba: Test del método modificar_expresion() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se modifica la expresión booleana con el parámetro introducido.
Operativa: Se crea una expresión, se modifica y se comprueba que se haya modificado correctamente.

BorrarExpresionTest()

Objeto de la prueba: Test del método borrar_expresion() de la clase DomainCtrl.
Valores estudiados: Se comprueba que se borra la expresión booleana con el parámetro introducido.
Operativa: Se crea una expresión, se borra y se comprueba que se haya borrado correctamente.

7. RELACIÓN IMPLEMENTACIÓN POR MIEMBRO EQUIPO

	Nombres			
	Hugo Aranda	Alejandra Gallardo	Eric Riera	Albert Ruiz
Clases	ExpBooleana	DomainCtrl	Usuario	Corpus
	ExpBooleanaCtrl	Archivo	UsuarioCtrl	VectorMEV
	GestorBooleana	CjtAutores	ArchivoCtrl	
	GestorIO	Formato	VistaPrincipal	Pair
	ParserPROP	PresentationCtrl	VistaRegistro	GestorAutor
	ParserXML	MenuCtrl	VistaCrearArchivo	GestorCorpus
	GestorArchivo	VistaMenu	VistaVisualizador	GestorUsuario
	ParserTXT	VistaComparador	VistaEditor	ParserXML
		VistaQuery	VistaPerfil	GestorArchivo
		VistaAutores		
		VistaExpBooleana		
		VistaDialogo		
Tests	ExpBooleanaTest	ArchivoTest	UsuarioTest	CorpusTest
	ExpBooleanaCtrlTest	CjtAutoresTest	UsuarioCtrlTest	VectorMEVTest
		DomainCtrlTest		
			ArchivoCrtlTest	
Drivers	DriverExpBooleana	DriverCjtAutores	DriverUsuarioCtrl	DriverArchivoCtrl
	DriverExpBooleanaCtrl	DriverCorpus		DriverVectorMEV
		DriverDomainCtrl		