UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

INFORME DE LABORATORIO 3: SIMULACIÓN DE UNA PLATAFORMA ESTILO "GOOGLE DOCS" EN JAVA

Nombre: John Serrano C.
Profesor: Roberto Gonzales I.
Asignatura: Paradigmas de Programación

24 de enero 2022

Tabla de Contenidos

1. INTRODUCCIÓN

El presente informe corresponde al laboratorio número 3 del curso de Paradigmas de Programación. Este tercer laboratorio corresponde al Paradigma Orientado a Objetos, donde se utiliza el lenguaje de programación JAVA a través de algún IDE a elección, como por ejemplo InteliJ IDEA. El informe sigue la siguiente estructura: Primero, se tiene una descripción breve del problema, luego una descripción del paradigma, los objetivos del proyecto, análisis del problema, diseño de la solución del problema, aspectos de implementación, instrucciones y ejemplos de uso, resultados y autoevaluación y finalmente una conclusión al proyecto realizado.

1.1 DESCRIPCIÓN DEL PROBLEMA

Se pide desarrollar una simulación de un editor de texto colaborativo, como, por ejemplo, Google Docs. Este editor permite registrar usuarios y estos al loguearse correctamente tienen la capacidad para crear un documento junto con su texto, editar el mismo y compartir el documento con otros usuarios registrados en el editor, entre otras cosas. Para implementar esto, se deben tener algunos elementos en cuenta:

Usuarios: Corresponden a los usuarios registrados en el editor. Cada usuario tiene un username y un password, que funcionan como las credenciales de este, junto con su fecha de registro en el editor. Los usuarios registrados y logueados correctamente pueden hacer todas las operaciones disponibles en el editor.

Documentos: Estos son creados por los usuarios y registran a un autor, una fecha de creación y un nombre, junto con el contenido principal del documento. Estos pueden ser compartidos con otros usuarios y todos los cambios del contenido del documento quedan registrados en el historial de versiones. Tienen un ID único.

Funcionalidades: Son aquellas que registran y loguean a un usuario y crean, comparten, editan y modifican a un documento en específico, entre otras cosas.

Todo debe ser implementado utilizando el Paradigma Orientado a Objetos, a través de la programación en Java.

1.2 DESCRIPCIÓN DEL PARADIGMA

El Paradigma Orientado a Objetos se basa en la definición de objetos, es decir, abstracciones de entidades que pueden tener atributos y métodos, con el fin de interactuar entre si (intercambio de información). Existen objetos de distintas clases, donde sus características y comportamientos pueden variar. Los comportamientos pueden leer o escribir las características del mismo objeto, por lo que pueden permitir interacciones entre sí.

Uno de los lenguajes mas populares en el ámbito de la programación orientada a objetos es Java, ya que es un lenguaje que trabaja fuertemente con los conceptos de clases, métodos y atributos.

En el Paradigma Orientado a Objetos se tienen algunos elementos bastante importantes a la hora de construir un código utilizando este paradigma, como, por ejemplo:

Clases: Son la definición de las características de un objeto las cuales tienen atributos y métodos. Una clase corresponde a una implementación de un TDA.

Objetos: Son instancias de una clase, es decir, representaciones activas de estas.

Atributos: Corresponde a lo que compone a una clase, como son distintos tipos de datos. También pueden ser otras clases.

Métodos: Son los comportamientos de los objetos. En palabras simples, son las "funciones" que componen a las clases. Los métodos expresan comportamientos que pueden realizar un objeto sobre si mismo o sobre otros objetos.

Para crear un objeto, se debe hacer uso de un constructor, el cual es un tipo especial de método. Permite indicar los valores iniciales de sus atributos y tiene la misión de reservar la memoria necesaria para poder albergar todos los datos del objeto.

Algo también importante mencionar es que se tienen distintos tipos de diagramas que permiten organizar de mejor forma las ideas sobre las clases, atributos y métodos en un código donde se utilizan varias clases. El principal diagrama UML es el diagrama de clase, el cual muestra las relaciones que existen entre las distintas clases que componen un código.

Además, también es posible implementar el Paradigma Orientado a Objetos utilizando el patrón MVC, donde principalmente las clases corresponden a los modelos. Para este proyecto, se utilizó el MVC, utilizando las partes de Modelo, Vista y Controlador.

1.3 OBJETIVOS

Como objetivos del proyecto se tiene aprender sobre el Paradigma y la programación Orientada a Objetos, para así obtener la habilidad de programar de otra forma distinta a la que se tiene costumbre actualmente. Otro objetivo es programar correctamente en Java y aprender a utilizar las herramientas de este lenguaje para completar el proyecto de laboratorio y así poder tener una base para futuros proyectos que se desarrollen con la Programación Orientada a Objetos.

2. DESARROLLO

2.1 ANÁLISIS DEL PROBLEMA

Cabe destacar de que en el proyecto no se trabaja con Bases de Datos o similares, ya que son conceptos que aún no se han introducido y, por lo tanto, solo se requiere realizar una simulación de un editor de texto colaborativo a través de la consola de comandos (CMD). Entendiendo esto, se tienen algunos elementos fundamentales en el problema:

Primero, el **menú,** que es la vía por la cual el usuario interactúa con el programa. El menú consiste en un conjunto de funcionalidades y debe permitir interactuar tanto con el editor como con las funcionalidades.

Segundo, el editor en sí, el cual es la base de todo y donde se guardan todos los demás elementos importantes, como son los usuarios, documentos, permisos, versiones de los documentos, entre otras cosas. El editor registra todos los cambios que se realizan exitosamente a través de una funcionalidad. Teniendo en cuenta esto, se requiere representar el editor en Java, donde se puede representar de la siguiente manera:

Editor: Objeto que contiene a un string, una fecha, un usuario activo y dos listas donde se irán guardando los usuarios registrados y los documentos respectivamente. (string X fecha X user X list X list).

A diferencia del primer laboratorio y al igual que el segundo, no es necesario trabajar con funciones de encriptación para el texto de las contraseñas y documentos.

Luego, se tienen otros elementos importantes que se guardan en el editor:

- **Usuarios:** Corresponden a aquellos que interactúan con el editor, registrándose en este para luego poder realizar las operaciones disponibles.
- **Documentos:** Son creados por los usuarios y tienen información escrita registrada en el documento. Los documentos se guardan en el editor y contienen información de identificación al igual que un ID único para poder diferenciar un documento de otro.
- **Permisos:** Corresponde a un conjunto de asociaciones, donde a un usuario se le da un grado de acceso a un documento. Existen tres tipos de permisos: Lectura, Escritura y Comentarios.
- **Historial de versiones:** Es donde queda registro de las versiones de un documento. Una versión nueva se produce cuando se realiza algún cambio en un documento, ya sea editar texto, eliminar texto, cambiar texto, etc. Cada versión tiene identificación. El historial de versiones de un documento se guarda en el propio documento.

Para poder implementar a los elementos mencionados anteriormente en Java, se pueden representar de la siguiente manera:

- **Usuarios:** Objeto que contiene una fecha, dos strings y un integer (fecha X string X string X integer). Cabe destacar que, para el caso del usuario activo, se guarda fuera de la lista de usuarios, como otro atributo aparte del editor, junto con un booleano que indica si está conectado o no.
- **Documentos:** Objeto que contiene contiene tres strings, una fecha, dos listas y un integer (string X fecha X string X string X list X list X integer)
- **Permisos:** Objeto que contiene 2 strings (string X string)
- **Historial de versiones:** Objeto que contiene una fecha, un string y un integer (fecha X string X integer)

Por último, se tienen las operaciones que se pueden realizar en la plataforma:

- **Authentication:** Verifica si un username no ha sido registrado. Si es así, registra al usuario en el editor. Además, permite verificar las credenciales de un usuario registrado y si son correctas, deja al usuario como usuario activo (logueo correcto). También permite al user desloguearse del editor. Es necesario que un User este logueado para que las operaciones a continuación se muestren correctamente en el menú, a excepción de Visualize.
- **Create:** Crea un nuevo documento y lo guarda en el editor.
- **Share:** Permite compartir un documento con otros usuarios registrados en el editor. Los permisos se guardan en el documento especifico.
- **Add:** Añade texto al final de la versión activa de un documento.
- **Rollback:** Restaura una versión de un documento, guardada en el historial.
- **RevokeAccess:** Elimina todos los permisos externos otorgados en un documento.
- **Search:** Permite buscar documentos (propios o que se tenga permisos) que contengan un texto en específico. Muestra por consola algunos datos de los documentos donde la búsqueda haya tenido éxito.
- **Visualize:** Convierte el contenido de todo el Editor en un string y lo muestra por consola. Funciona tanto estando logueado como no logueado.
- **Delete:** Permite eliminar texto en una versión activa de un documento.
- **SearchAndReplace:** Permite buscar y reemplazar un texto en una versión activa de un documento.

Cabe mencionar que todas las opciones anteriores están disponibles en el menú interactivo, mostrándose, dependiendo si el usuario este logueado o no.

Se pide que cada archivo corresponda a una clase, donde se utilice correctamente la documentación en estilo Javadoc para documentar métodos y clases. Además, también se solicitan los diagramas UML de clases antes y después de haber hecho el proceso de solución.

2.2 DISEÑO DE LA SOLUCIÓN

Para diseñar la solución, no solo es necesario utilizar los tipos de datos nativos de Java, sino que también se deben crear tipos de datos específicos, a través de lo que se conoce como TDA, pero como estamos trabajando con el Paradigma Orientado a Objetos, a los TDAs se les conoce como "Clases". Cabe destacar de que en a pesar de que era opcional, se decidió utilizar el Modelo Vista Controlador (MVC) para poder organizar mejor el código, junto con otros beneficios mas que trae utilizar este.

Las clases creadas son utilizadas para la construcción de las operaciones de otras clases y para las funcionalidades obligatorias y opcionales. Las clases más importantes son:

2.2.1.1 CLASE EDITOR

- Representación: Contiene un string, una fecha, una lista para guardar a los usuarios registrados, una lista para guardar a los documentos, un usuario el cual corresponde al usuario actualmente activo y un booleano para saber si el activo está conectado.
- Constructor: Dado un nombre, se crea un nuevo editor el cual ya contiene una fecha de creación determinada, gracias a la utilización de java.util.Date, la cual permite obtener automáticamente la fecha actual. También tiene dos listas vacías, un usuario inicializado en null y un booleano inicializado en false
- Selectores, Modificadores y otros Métodos: *Ver Figura N°2 en Anexos, P. 13.*

2.2.1.2 CLASE USUARIO

- Representación: Contiene una fecha, dos strings y un integer.
- Constructor: Dados un username y un password, se crea un objeto que contiene estos dos elementos junto a una fecha de creación y un id.
- Selectores, Modificadores y otros Métodos: *Ver Figura N°2 en Anexos, P. 13.*

2.2.1.3 CLASE DOCUMENTO

- Representación: Contiene tres strings, una fecha, dos listas y un integer
- Constructor: Dados un autor, un nombre y un texto, se crea un objeto que contiene las entradas anteriormente mencionadas junto con las dos listas, la fecha y el id global. La primera lista corresponde a los Permisos y la segunda lista corresponde a un Historial de Versiones.
- Selectores, Modificadores y otros Métodos: *Ver Figura N°2 en Anexos, P.13*

2.2.1.4 OTRAS CLASES

Las clases más "pequeñas", pero no menos importantes son:

- **Clase Historial:** Representado como un objeto que contiene una fecha, un string y un integer.
- **Clase Permisos:** Representado como un objeto que contiene dos strings.

- **Clase Controlador:** Representado como un objeto que contiene un Editor.
- **Clase Menú:** Representado como un objeto que contiene al controlador.

Tanto antes del proceso de desarrollo como después se realizó un diagrama de análisis y un diagrama de diseño. Estos fueron realizados colocando las clases, métodos y atributos considerados tanto antes como después del proceso de solución. *Para ver ambos diagramas, ver en Anexos, Página 13.*

2.2.2 FUNCIONALIDADES OBLIGATORIAS

AUTHENTICATION: Se implementa un método el cual permite verificar si un usuario está registrado en el editor y si no es así, se registra al Usuario. (Register). Además, permite loguear a un Usuario si corresponde a uno de los Usuarios registrados en el editor (login) como también eliminar la sesión activa (logout). Si el login es exitoso, el menú se actualiza y muestra nuevas funcionalidades para aplicar.

CREATE: Se implementa un método que crea un Documento cuyo autor corresponde al Usuario activo. Se agrega a la lista de documentos del Editor. La gran mayoría de las opciones en el menú del editor requieren de un documento creado anteriormente.

SHARE: Se implementa un método, donde se recibe una lista de usuarios y un permiso. Se verifica que el ID corresponda a uno de los documentos del Editor. Se filtran los usuarios, dejando solo a los registrados en el Editor y concede el permiso correspondiente, dependiendo del permiso ingresado a través del menú.

ADD: Se implementa un método donde se verifica que el ID corresponda a uno de los documentos del Editor. Además, se verifica que el usuario activo sea el autor del documento o bien tenga permisos para editar. Se agrega el texto al final de la versión activa, creando una versión nueva y la antigua queda guardada en el historial.

ROLLBACK: Se implementa un método donde se verifica que el ID corresponda a uno de los documentos del Editor. Luego se verifica que el ID del historial corresponda a una de las versiones y se reemplaza la versión activa del Documento por una antigua del Historial. Se elimina la sesión activa del User.

REVOKEACCESS: Se implementa un método donde se verifica que el ID corresponda a uno de los documentos del Editor, creando una nueva lista de permisos vacía, se reemplaza utilizando los setters por la lista actual, eliminando así todos los permisos externos.

SEARCH: Se implementa un método que a través del uso del método nativo .contains() busca en todos los documentos donde el usuario activo sea el dueño o tenga permisos y que contengan un texto en específico. Retorna por consola algunos datos importantes de los documentos donde la búsqueda tuvo éxito, como el nombre, ID, y autor.

VISUALIZE: Se implementa un método donde dependiendo de si hay un usuario logueado, se transforma la información del usuario a string (su username, fecha de registro y documentos a los cuales puede acceder). Si no hay usuario logueado entonces se muestra por consola la información de todos los documentos creados en el Editor, como, por ejemplo, autor, nombre, cantidad de versiones, versión activa e ID.

2.2.3 FUNCIONALIDADES OPCIONALES

DELETE: Se implementa un método que a través del uso del método nativo .substring() elimina una cantidad de caracteres en específico de la versión activa de un documento. En el caso donde la cantidad de caracteres a eliminar sea mayor al largo de la versión actual, simplemente se deja la versión activa como “”.

SEARCHANDREPLACE: Se implementa un predicado que utiliza el procedimientos de Search junto con el predicado nativo .replaceall(), donde primero se utiliza .contains() para asegurarse de que el string a reemplazar este en el documento y luego se utiliza .replaceall() para reemplazar todas las coincidencias de la palabra a reemplazar.

Cabe destacar que, por temas de tiempo, solo se implementaron 2/4 funcionalidades opcionales.

2.3 ASPECTOS DE IMPLEMENTACIÓN

2.3.1 COMPILADOR / IDE

Para este proyecto es necesario utilizar algún IDE de Java, como, por ejemplo, Netbeans o inteliJ IDEA. Para este proyecto específicamente, se utilizó InteliJ IDEA en su versión 2021.3.1 (Ultimate Edition). En el caso de JDK, se utilizó la versión 11.0.13.8 para compilar el programa. Se utilizó solo funcionalidades pertenecientes a la librería estándar de Java y el programa fue probado y compilado en el sistema operativo Windows 10.

2.3.2 ESTRUCTURA DEL CÓDIGO

Como se utilizó el patrón MVC, el código fuente tiene cuatro carpetas: controller, view, model y main. La carpeta model contiene a todas las clases correspondientes a los “TDAs”, como son Editor, Usuario, Documento, Permiso e Historial. En la carpeta controller está la clase donde se desarrollan las funcionalidades obligatorias y opcionales. En la carpeta view está la clase del Menú interactivo, donde se desarrolló todo el código para la ejecución de este. Finalmente, en la carpeta main se tiene a la clase principal del programa, donde se utiliza la clase controlador y la clase menú, importándolas desde sus respectivos paquetes.

Al contrario de los laboratorios anteriores, no se deben incluir ejemplos ya que todas las funcionalidades se pueden probar a través del menú, pero por defecto se tienen a 5 usuarios registrados en el editor y 10 documentos creados en el editor. Cada método y clase se comentó utilizando Javadoc.

2.4 INSTRUCCIONES DE USO

2.4.1 EJEMPLOS DE USO

Lo primero que se debe verificar es si tenemos JDK / JRE instalado en su versión 11. Luego, se debe verificar que se tenga la carpeta SRC del proyecto, con las cuatro carpetas descritas anteriormente. Dentro de la carpeta SRC se tiene un archivo .bat llamado "script.bat". Se puede ejecutar el archivo .bat en el CMD (estando en la ruta de la carpeta SRC y escribiendo el nombre del archivo) o bien, se puede hacer doble click en el archivo para iniciar el archivo y aparecerá el menú interactivo en pantalla. Luego es cosa de seguir las instrucciones que se muestran en pantalla.

Se tienen distintas opciones dependiendo si se está logueado en el editor o no. Si no se está logueado, se puede registrar, loguear, ver los documentos creados en el editor o cerrar el programa. Si se está logueado, se pueden crear nuevos documentos, compartir un documento, añadir texto al final de un documento, restaurar una versión de un documento, eliminar todos los permisos de un documento, buscar en documentos, ver los datos del usuario activo, borrar caracteres de un documento, buscar y reemplazar en un documento, desloguearse y cerrar el programa. El menú da distintos feedbacks dependiendo de las opciones ingresadas o de las cosas que se deban ingresar dependiendo de las funcionalidades.

Para ver algunas pruebas con el menú interactivo, ver en Anexos Figuras 3, 4 y 5 en las páginas 14, 15 y 16.

2.4.2 RESULTADOS ESPERADOS

Se espera crear una simulación de un editor de texto colaborativo, donde el programa sea funcional en su totalidad, sin errores ni ambigüedades.

Junto con lo anterior, se espera trabajar correctamente con clases y métodos, ya que son fundamentales para el desarrollo del proyecto. Por último, se espera que cada método haga su procedimiento correctamente, que las clases tengan representaciones correctas y que al ejecutar el programa se tenga un menú completamente funcional que permita utilizar las distintas funcionalidades implementadas en el editor.

2.4.3 POSIBLES ERRORES

Como tal, no se tienen errores ya que en el caso de que el usuario decida ingresar una opción no valida en el editor, este da feedback sobre lo que se ingresó erróneamente o lo que se debe ingresar. Por lo tanto, si se siguen todas las instrucciones del menú interactivo no habrá errores ni inconvenientes con el programa.

2.5 RESULTADOS Y AUTOEVALUACIÓN

2.5.1 RESULTADOS OBTENIDOS

Los resultados obtenidos fueron los esperados, ya que se logró crear todas las funcionalidades obligatorias y la mitad de las opcionales. El programa es completamente funcional, incluyendo en casos de contraejemplos donde las entradas colocadas son erróneas o donde se decide ingresar otro número que no corresponde a ninguna opción en el menú. Por lo tanto, se logró crear una simulación de un editor de texto colaborativo.

Se hicieron múltiples pruebas con distintos ejemplos para probar de que no hubiera fallos en la ejecución del código y que el código hiciera lo correcto. Se probaron varios casos, considerando los distintos grados de accesos que los usuarios pueden tener en los documentos y además también se probaron algunos casos “erróneos” donde, por ejemplo, el usuario decide ingresar otra entrada totalmente contraria a la sugerida por el menú interactivo. En este caso, no se encontraron errores ya que el menú verifica que las entradas sean las correctas y en caso contrario retorna un mensaje mencionando que se ingresó incorrectamente.

2.5.2 AUTOEVALUACIÓN

La Autoevaluación se realiza de la siguiente forma: 0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces.

Debido a que se probaron las funcionalidades con varios casos y no se encontraron errores, se considera que todas las funcionalidades implementadas funcionan el 100% de las veces.

Para ver la tabla de Autoevaluación, ver la Tabla N°1 en Anexos, P.17.

3. CONCLUSIÓN

Luego de realizar y completar el proyecto, se puede concluir que se cumplieron los objetivos principales, ya que fue posible aprender a utilizar correctamente Java para poder completar el proyecto de laboratorio correspondiente. Se obtuvo aprendizaje sobre un paradigma nuevo y se pudo aplicar correctamente, utilizando los conceptos vistos en clase como son las clases, métodos, atributos, constructores, paso de mensajes, composición, entre otros.

Una de las complicaciones más grande para la realización del proyecto, fue que a veces algunas ideas no resultaban bien del todo y posterior a ver el resultado de una idea se decidía cambiarla si es que era necesario. Esto para evitar errores o facilitar la lectura del código, como, por ejemplo, al principio no se tenía planeado utilizar el MVC, pero al ver que funciona como una forma de organizar mejor el código, junto con querer aprender a utilizar cosas nuevas, se decidió utilizarlo en el proyecto, a pesar de que es opcional. Otra gran complicación fue el tiempo, ya que este es un proyecto que requiere de bastante tiempo para su comprensión y para lograr realizar todo lo requerido y lamentablemente esta vez no se pudieron implementar todas las funcionalidades opcionales, pero, aun así, se pudo completar el proyecto en su gran mayoría y obtener un programa completamente funcional con distintas funcionalidades implementadas.

Dejando lo anterior de lado, no hubo complicaciones de comprensión de conceptos o problemas con las herramientas como Git o Java en general. Al contrario, gracias a este proyecto se pudieron aprender muchas herramientas y métodos nativos de Java, algo que ayudó mucho también a la hora de preparar evaluaciones.

Comparado con los paradigmas anteriores, los cuales eran el Paradigma Funcional y el Paradigma Lógico, en el paradigma Orientado a Objetos resultó un código mucho mas corto y simple, incluso más simple que en Prolog, debido a la opción de crear automáticamente los constructores, getters y setters con el IDE InteliJ IDEA, lo que también provoca que se ahorren muchas líneas de código, algo que ni en Scheme ni en Prolog sucedía ya que se debían construir las operaciones desde cero. Personalmente, considero que el Paradigma Orientado a Objetos es un paradigma que tiene bastantes cualidades interesantes y que permite realizar muchas cosas más comparado con Scheme y Prolog, además de su simplicidad a la hora de crear los TDAs, que en este caso son las clases.

Como este es el laboratorio final del semestre, se puede concluir que se logró utilizar completamente el concepto de paradigma, y se espera que este conocimiento sirva no solo para posteriores asignaturas, si no para cuando ya se esté trabajando en la industria y se requiera introducirse en distintos lenguajes y paradigmas de programación.

4. BIBLIOGRAFÍA Y REFERENCIAS

1. Gacitúa, D. (2021 - 2022). “Proyecto Semestral de Laboratorio”. Paradigmas de Programación. Enunciado de Proyecto Online. Recuperado de: https://docs.google.com/document/d/1pORdJwp5WJ7V3DIAQik9B58llpdxpurQRVTKPZHOpS8/edit
2. Gacitúa, D. (2021 - 2022). “5 - P. Orientado a Objetos”. Paradigmas de Programación. Material de clases Online. Recuperado de: https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=20
3. Chacon, S. y Straub, B. (2020). “Pro Git – Todo lo que necesitas saber sobre Git’’. Libro Online. Recuperado de : https://drive.google.com/file/d/1mHJsfvGCYcIhdrmK-lBI6a1WS-U1AAPi/view
4. Multiples Autores. (2007). “Object-Oriented Analysis & Design”. Libro online. Recuperado de: https://drive.google.com/file/d/1f9yqn9FTNtPGMX0Yc11PtpyS7KIv322X/view?usp=sharing

5. ANEXOS

Figura N°1: Diagrama de análisis UML de clases

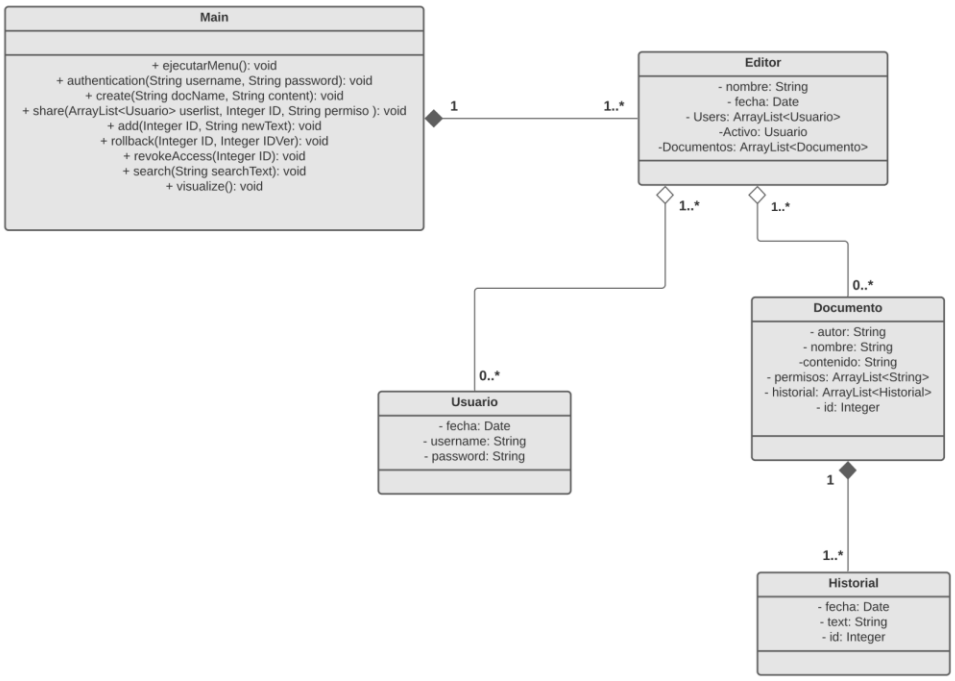

Figura N°2: Diagrama de diseño UML de clases

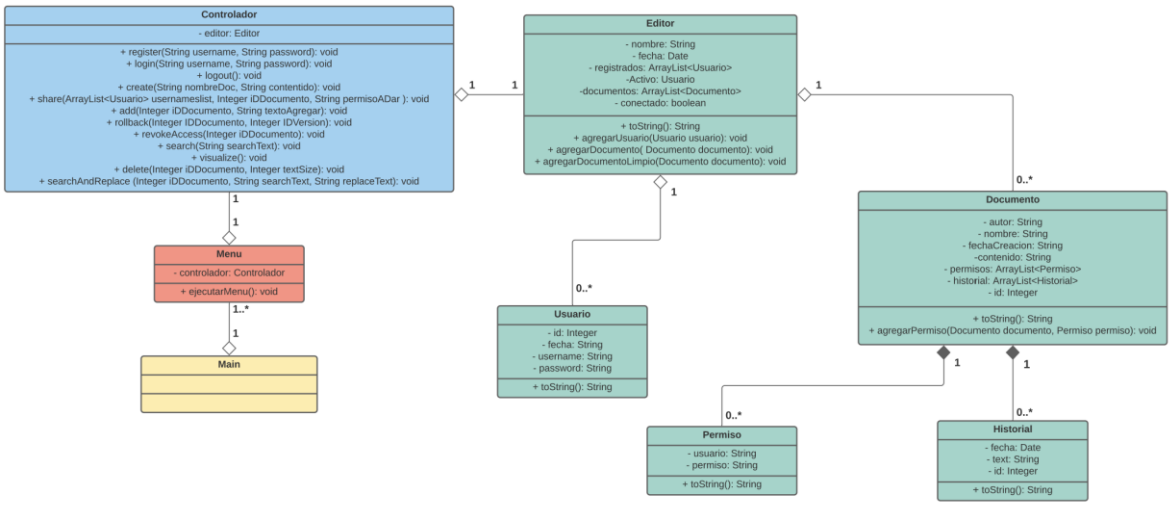

Figura N°3: Seleccionando Visualize sin estar logueado.

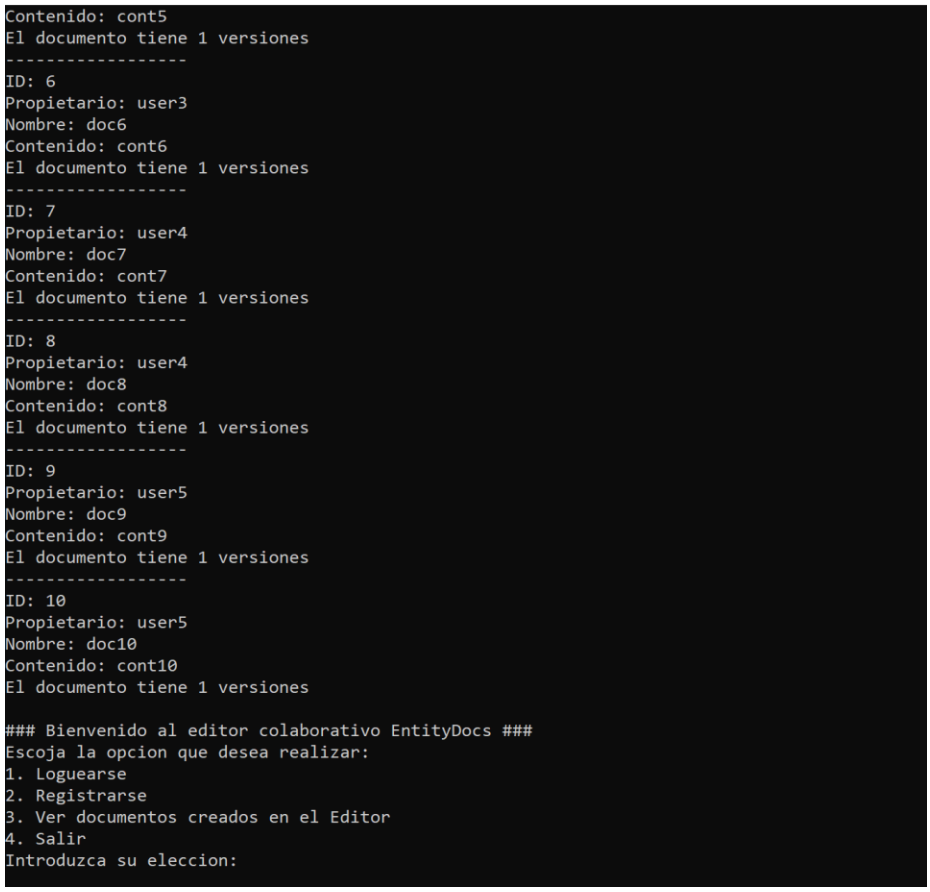

Figura N°4: Logueado en el menú interactivo

```
### Bienvenido al editor colaborativo EntityDocs ###
Escoja la opcion que desea realizar:
1. Loguearse
2. Registrarse
3. Ver documentos creados en el Editor
4. Salir
Introduzca su eleccion:
1
Su opcion fue la numero 1: Loguearse
Ingrese el nombre de usuario:
user1
Ingrese la contrasena para el usuario user1
pass1
### Editor EntityDocs ###
## Logueado como: user1 ##
Escoja una accion a realizar:
1. Crear nuevo documento
2. Compartir documento
3. Agregar contenido a un documento
4. Restaurar version de un documento
5. Revocar acceso a un documento
6. Buscar un texto en los documentos
7. Visualizar documentos
8. Eliminar caracteres de un documento
9. Buscar y reemplazar un texto de un documento
10. Cerrar sesion
11. Cerrar el editor
Ingrese una de las opciones anteriores:
```

Figura N°5: Seleccionando Visualize estando logueado como User1 (Este user ya viene cargado inicialmente)

```
Ingrese una de las opciones anteriores:
7
Nombre del editor: EntityDocs
Fecha de creacion de la plataforma: 24/01/2022
------------------
Usuario online actualmente: user1
Cuenta creada con la fecha de: 24/01/2022
------------------
El usuario cuenta con 3 documentos propios o compartidos
------------------
ID: 1
Propietario: user1
Nombre: doc1
Contenido: cont1
El documento no ha sido compartido con otros usuarios
El documento tiene 1 versiones
------------------
ID: 2
Propietario: user1
Nombre: doc2
Contenido: cont2
El documento no ha sido compartido con otros usuarios
El documento tiene 1 versiones
------------------
```

Tabla N°1: Autoevaluación de los Requerimientos Funcionales

Requerimientos Funcionales	Evaluación
Clases y estructuras que forman el programa	1
Menú interactivo por terminal	1
Authentication	1
Create	1
Share	1
Add	1
Rollback	1
RevokeAccess	1
Search	1
Visualize	1
Delete	1
SearchAndReplace	1