Universidad de Santiago de Chile

Paradigmas de la Programación (13310-0-A-1)

Informe 2 Paradigma Lógico

Docente: Roberto González Ibáñez

Estudiante: Nícolas Farfán Cheneaux

Enero 2022

Contenido

Introducción

El siguiente paradigma de esta asignatura corresponde al lógico, el cual se basa en predicados y definiciones de reglas lógicas, perteneciente a la familia de paradigmas declarativo, sobre este se construirá una plataforma de documentos de ofimática colaborativa estilo *Google Docs*, el propio contará con las funcionalidades básicas de este, a continuación se describen las características de estas y como se implementan en un lenguaje de programación lógica llamado Prolog, por sus siglas en francés PROgramation en LOGiquem es un lenguaje seminterpretado, el código se compila a código de byte mediante una máquina virtual Warren Abstract Machine(WAM).

Descripción del problema

Las plataformas de documentos colaborativas tienen múltiples usos y aplicaciones actualmente, sus operaciones principales abarcan desde lo simple, *crear* y *escribir* un documento propio a lo más complejo editar un documento compartido junto con otros usuario, es posible añadir texto, eliminarlo y cada vez que se realice un cambio queda registrado en el historial de versiones como una nueva versión, y el cambio realizado a la última versión o activa se convierte en la nueva versión activa, además sobre los documentos existen permisos (escritura, lectura, comentarios y compartir), para más información ver diagrama(Anexo 1), además sobre los documentos es posible buscar texto en versiones a las cuales se tenga acceso, es decir un usuario anteriormente compartió su documento con este, además se permite revocar los accesos a los usuarios que anteriormente se compartió.

Descripción y uso del paradigma

El paradigma lógico pertenece a la familia de lenguajes del paradigma declarativo, y está basado en la lógica simbólica, por lo tanto está orientado a las metas, por lo tanto, el estilo de programación se basa en especificar que se debe lograr y no en cómo.

Al momento de realizar un programa en prolog, se definen hechos, relaciones y reglas, las cuales la base de conocimientos de Prolog asume como conocidos, para luego en base a estas realizar consultas.

- **Hechos en Prolog**: Son usados para definir en la base de conocimiento que conjunto de individuos tienen tal característica, de la forma: mamifero(perro), leyéndose el perro es un mamífero, nótese que esta afirmación no depende de otra para que sea verdadera.
- **Reglas en Prolog:** Es una relación de implicacia, que tiene un consecuente que será verdadero siempre y cuando los antecedentes sean verdaderos por ejemplo: animal(X):-mamifero(X), leyéndose de la forma: X es un animal si X es un mamífero.
- **Consultas:** Como definimos una variable “X” sobre esta se pueden hacer consultas y prolog determinará si son true/false si se satisface en su base de conocimiento.

En prolog existen diversos tipos de dato, uno de los más usados son las variables anónimas, ya que nos permite no tormala en cuenta en una cláusula, ya que nuestro predicado no depende de esta para que determine si es true/false.

Como el presente proyecto está basado en listas, prolog representa una lista de forma recursiva, compuesta por una cabeza y una cola:

Lista = Nulo | Cabeza X Lista

En un predicado se vería: **predicado[Head | Tail],** la cabeza separada de la cola por el operador " | " pipe, el cual nos permite descomponer la lista y más adelante crear predicados recursivos sobre estas.

Conceptos Usados en este proyecto:

La recursividad es usada en todos los predicados para recorrer, filtrar, añadir elementos sin duplicados en este proyecto, ya que nuestro proyecto está basado en extraer los datos de un documento o elemento y los demás no se deben ver involucrados, por lo tanto se construyen nuevamente con el elemento modificado y los demás sin ningún cambio, para estas operaciones se crearon predicados propios, ya que se debe utilizar el operador de corte para evitar obtener otras soluciones, ya que en este proyecto no hay una base conocimiento para tener distintas soluciones.

Las variables anónimas también son usadas para obtener un elemento específico de una lista, sin que influyan los demás.

Análisis del Problema

La presente plataforma de documentos presentada en los requerimientos funciona con 2 elementos particulares: Los **usuarios** y los **documentos,** en este sentido podemos desglosar el problema en 2 entidades fundamentales, donde la primera de estas tiene el control sobre el segundo, pudiendo crear, editar, otorgar acceso, sobre este.

Por otra parte los documentos tienen su propia estructura y un identificador clave, cada documento es único y puede ser usado al mismo tiempo por otros usuarios (En términos de edición y creación de versiones).

Datos de entrada:

El usuario requiere una autenticación previa para ingresar y usar las características de la plataforma, tiene un conjunto de operaciones que puede hacer simplemente estando registrado en esta, caso contrario no puede acceder a ninguna herramienta.

Por otra parte, no es necesario que el usuario tenga un documento para empezar a editar o crear, puede ser colaborador de otro documento previamente creado por otro usuario.
Los permisos que un usuario puede llegar a tener son: ***Escritura, Comentarios, Lectura, Compartir***

Los documentos tienen una estructura particular la cual se rige por lo requerido:

Id xx DueñoDoc xx TítuloDoc xx Usuarios Con Acceso xx Historial de versiones, por ejemplo:

[1, "saul", "saulDoc", [["saul", ["W", "C", "R", "S"]]], [[0, [20, 12, 2020], "saulContent"]]]

Se lee de la siguiente forma: El documento de título "SaulDoc" tiene id:1, su creador es "Saul", ningun usuario tiene permisos, y tiene una sola versión (id 0).

Símbolos: W: escritura, C: comentarios, R: lectura, S:Compartir, (-): Cualquier Usuario Registrado

- Logearse **(-)**
- Crear Documento **(-)**
- Añadir Texto **(W)**
- Otorgar Permisos **(W,S)**
- Editar Permisos **(W,S)**
- Restaurar una versión **(-)**
- Revocar Accessos **(S)**
- Eliminar Texto **(W)**
- Buscar en Documentos **(-)**
- Reemplazar texto **(A, w)**

Una versión del historial de versiones se compone de la siguiente estructura:

Id Version, Fecha de Modificación Versión xx Contenido Versión xx, por ejemplo:

[[1, [21, 12, 2021], "kimContent Agrego 1 2 3 "], [0, [20, 12, 2020], "kimContent"]]

El presente historial tiene 2 versiones las cuales la primera es la versión activa id:1 y se lee de la siguiente forma, la versión 1 tiene fecha de modificación 21/12/2021 tiene contenido *KimContent Agrego 1 2 3.*

Datos de salida:

En términos de esta plataforma, el usuario debe logearse para ejecutar una operación, (A excepción de 1 operación), una vez correctamente logeado, verificado las credenciales puede acceder a sus documentos y realizar operaciones, cada vez que se ejecute una operación, esta retorna una vista completa actualizada de la plataforma con el cambio realizado en los documentos, nótese que solo es posible seleccionar una operación a ejecutar.

Diseño de la solución

El presente proyecto está basado en listas, por lo tanto se requiere el uso de reglas recursivas para el manejo de listas, es decir que el consecuente debe satisfacer la cabeza de una regla, este concepto se utilizará para todas las operaciones (predicados) que se implementan en el presente proyecto.

Operaciones de Registro / Logeo / Permisos de usuarios

El predicado Register, toma los 3 datos de registro para que el usuario se encuentre registrado / miembro de ParadigmaDocs, su username funciona como idUsuario ya que es único en paradigmadocs, esto quiere decir que no pueden existir 2 usuarios con el mismo username en la plataforma, la contraseña se utiliza para iniciar sesión y la fecha de registro, como su nombre indica, representa la fecha de creción de la cuenta.

Posterior a esto si se quiere ejecutar cualquier operación, se implementa un predicado **login** el cual tiene la función que el usuario inicie sesión con su **username y contraseña** los cuales deben coincidir con algún usuario registrado en la plataforma. Si el logeo es exitoso, el nombre del usuario se añade representativamente a paradigmadocs en una lista en específico por ejemplo:

Ningún usuario logeado = **[ ]** (Una lista vacia)

Algún usuario logeado = **["username del usuario logeado"]**

Una excepción que se debe considerar al momento de logear a un usuario es que debe logearse e instantáneamente después de logearse, debe ejecutar alguna otra operación para deslogearse y otro usuario pueda utilizar la plataforma y ejecutar alguna operación, es decir ParadigmaDocs no se puede utilizar en paralelo.

Operaciones sobre los documentos

Una vez la autenticación del usuario esté correcta, ahora este usuario está habilitado para realizar cualquier operación, siempre y cuando tenga los permisos adecuados, sin embargo existen operaciones las cuales no es necesario tener un permiso por ejemplo: **Create/paradigmaDocsToString,**

Creación de documentos:

Create: *(Ver Anexo 2)*

1) El usuario miembro de paradigmadocs esta autorizado de crear los documentos que el desee, identificado por un **id** único y un autor único, más adelante se añadirán colaboradores.

Operaciones de acceso a los documentos:

Share: *(Ver Anexo 3)*

1) El usuario ingresa una lista de permisos y una lista de usuarios registrados a los que se le quiere otorgar, tal permiso.
2) El resultado es añadir a tales usuarios a la lista de accesos, los cuales de ahora en adelante podrán realizar las operaciones que sus permisos le permitan hacer.

Revoke All Accesses: *(Ver Anexo 4)*

1) Contrario a Share, esta operación permite al dueño del documento eliminar todos los accesos que en un principio había otorgado mediante Share.
2) El resultado es eliminar todos los permisos de los documentos que el usuario prefiera.

Operaciones de edición documentos

Estas operaciones pueden ser realizadas por usuarios que tienen permiso de edición sobre los documentos, permiso "W" y los autores, estas son:

Add *(Ver Anexo 5)*

1) Recurre a la versión activa del documento y añade al final de la cadena de texto el nuevo texto a añadir.
2) Crea una nueva versión con el nuevo texto añadido.

Restore Version *(ver Anexo 6)*

1) Recurre al historial del documento mediante su id, busca una nueva versión, mediante su Id y cambia la versión activa por alguna escogida por el usuario.
2) El resultado es una nueva versión activa actualizada y la anterior versión se añade al historial.

Delete *(ver Anexo 7)*

1) Recurre a la versión activa del documento y elimina los últimos "n" caracteres ingresados por el usuario.
2) El resultado es una nueva versión creada por el cambio anterior de eliminar los "n" caracteres, la cual pasa a ser la versión activa.

Paradigmadocs –> String *(ver Anexo 8)*

1) Obtiene toda la información de paradigmadocs para convertirlo en String, predicados independientes ejectuan la transformación de cada TDA, (Documento, Versión, Acceso, Usuario)
2) Como el resultado anterior son listas de Strings, se transforma a string mediante predicados de concatenación de String.

Operaciones de búsqueda

Search *(ver Anexo 9)*

1) El usuario ingresa que carácter o palabra desea buscar en todos los documentos que tenga acceso (Sea dueño o se le haya sido compartido).
2) El resultado es todos los documentos en donde se haya encontrado una coincidencia.

Search and Replace *(ver Anexo 10)*

1) El usuario recurre a la versión activa de un documento (propio o con permiso de edición sobre este).
2) Ingresa el carácter o palabra buscada y si se encuentran coincidencias es reemplazado por otro carácter o palabra ingresado por el usuario
3) El resultado es una nueva versión activa producto del reemplazo de una palabra/carácter.

Consideraciones de Implementación

Estructura del proyecto

El proyecto se estructura en diferentes tipos de datos para representar a todos los elementos que conforman la plataforma, incluso la misma plataforma es un tipo de dato, en la que se almacenan los demás TDA's.

Estos se conforman en:

- TDA Usuario: Posee el constructor de la estructura de un usuario, representado por una lista: **[Username X Password X Date],** además se incluyen operaciones de verificación si un usuario ya se encuentra en la plataforma.

- TDA Documento: Posee el constructor de la estructura de un documento, representado por una lista: **[IdDoc XX Autor XX TituloDoc XX Permisos XX Versiones],** dentro del documento también existen 2 otros tipos de datos: Permisos o Accesses, Historial.

- TDA Permiso: Posee el constructor de la estructura de un permiso, y las operaciones de creación de esta, la cual viene asignada por 2 listas:

- Lista de Permisos de la forma: **[Permiso XX Permiso XX…]**
- Lista de usuarios permitidos: **[Username XX Username XX…]**

Y el constructor agrupa ambas listas, de la forma:

[[Username1 Permiso1][Username2 Permiso2]…]

además el presente TDA verifica que los permisos sean correctos y que los usuarios existan, para luego añadir los accesos al documento.

Por términos prácticos también se considera que el autor del documento automáticamente tenga permisos de escritura, lectura, comentarios, compartir; ya que al ser dueño, tiene poder absoluto sobre el documento, la lista de accesos quedando de la forma:

[[Autor ["W","C","R","S"],[Username1 Permiso1],[Username2 Permiso2]…]

- TDA Versión: Posee el constructor de la estructura de una versión del historial de documentos, de la forma:

[IdVersion XX FechaCambio XX ContenidoVersion]

Donde cada versión posee un **id** único e incremental, la fecha de creación de la versión y el contenido modificado, el historial es construido a partir de versiones, de la siguiente forma:

[Version1 XX Version2 XX Version3 XX …]

Este TDA construye todas las versiones y crea la versión activa de el documento que en términos prácticos es la primera versión del historial, de la forma:

[Versión Activa XX Versión A XX Versión B XX Versión C], nótese que no importa el orden interno del historial, solo importa la versión activa o cabeza de la lista.

Bibliotecas empleadas

Para este proyecto no se consideró el uso de ninguna biblioteca en particular.

Compilador o Interprete usado

SwiProlog 8.x.x, seminterpretado

Razones de la elección

Es requerimiento trabajar en este lenguaje y en su respectivo compilador.

Resultados Esperados

Se espera que el proyecto sea útil y cumpla con los requerimientos de una plataforma de documentos.

Posibles Errores

Si se ejecutan operaciones que no están definidas o previamente declaradas, o se ingresen tipos de datos distintos a los permitidos.

Resultados y Autoevaluación

Finalmente probando el proyecto con los ejemplos provistos, obtenemos una plataforma completa con todas las “operations” funcionando correctamente.

Evaluación

Requerimiento	Descripción	Resultado
TDA’s	Los nuevos tipos de datos creados cumplen su función y forman parte del resultado final.	
Register	Registrar a cualquier nuevo usuario que quiera utilizar ParadigmaDocs.	
Login	Permite iniciar sesión a cualquier usuario previamente registrado y queda habilitado para realizar cualquier operación.	
Create	Permite al usuario crear documentos.	
Share	Permite al usuario compartir documentos con otros usuarios de la plataforma.	
Add	Permite al usuario añadir texto a los documentos, los cuales es dueño o posee permiso de edición.	
Restore Version	Permite al usuario restaurar una versión anterior no activa del documento y convertirla ahora en una nueva versión activa.	
Paradigmadocs->String	Permite visualizar ParadigmaDocs de una forma clara, entendible para el usuario.	

Delete	Permite al usuario eliminar contenido de la versión activa de los documentos los cuales dueño o posee permiso de edición.	
Revoke All Accesses	Permite al propietario de un documento eliminar todos los accesos anteriormente otorgados	
Search	Permite al usuario buscar texto en cualquier documento (tanto en la versión activa como en el historial), siempre y cuando tenga algún tipo de permiso o sea propietario.	
Search and Replace	Permite buscar texto en la versión activa del documento, siempre y cuando tenga permiso de edición o sea propietario, para luego reemplazarlo por nuevo texto ingresado por este.	

Conclusión

Alcances

La programación declarativa y particularmente la programación lógica hacen que la ejecución de un proyecto basada en listas sea más sencillo al basarse en el *Qué* más no en el cómo, la programación lógica solo utiliza los hechos o predicados que el programador declara, es decir si alguno no existe, cualquier consulta sobre este será falsa, las variables anónimas también cumplen esté propósito al preocuparse solo de las variables que influyen en tal predicado, se entregarán un conjunto de soluciones, la definición de TDA es más práctico en este paradigma al incluir todo dentro del constructor, además las operaciones o predicados pedidos no requieren declarar variables ni recorrer una lista manualmente sino que con la recursividad y solo declarar lo que es verdadero y falso, con poco código basta para realizar un predicado requeridos.

Limitaciones

Entender cómo funciona la recursividad y el manejo de listas en un lenguaje con representación distinta fue una limitante al empezar el proyecto, ya que no se tenía la experiencia manejando listas y haciendo operaciones básicas sobre estas, además el modificar elementos dentro de estas y filtrar es difícil de entender a la primera, pero gracias al anterior laboratorio donde se voy programación declarativa se pudo afrontar con mayor facilidad estos conceptos.

Contraste Paradigma Anterior (Funcional)

En el pasado proyecto, realizado en el paradigma funcional, en el lenguaje Scheme se utilizaba un enfoque también declarativo, en prolog se utiliza la programación declarativa pero más a fondo, declarando lo que es verdad mediante hechos y reglas que satisfagan talas hechos, evitando el uso de sentencias condicionales como **if** o **else,** en términos de facilidad y sintaxis, en prolog no es necesario aprender sintaxis nueva al ser muy cortos los caracteres especiales propios del lenguaje que se ocupan, por el lado de las listas, ambos utilizan el concepto de una lista recursiva, formada por una cabeza y una cola, la programación lógica permite a partir de hechos construir una lógica mayor, teniendo definido lo que es verdad, en el paradigma funcional de Scheme se utilizaban funciones y composición de estas, además de las funciones anónimas, en el paradigma lógico solo se utilizan hechos y predicados.

Referencias

- (2004) El lenguaje de programación Prolog,

https://uvirtual.usach.cl/moodle/pluginfile.php/369182/mod_resource/content/1/PracticasPROLOG%20con%20apuntes.pdf

- (2002, February 7). lenguaje de programación. Retrieved December:

https://es.wikipedia.org/wiki/Prolog

-Manual. (2016) from Swiprolog.org website:

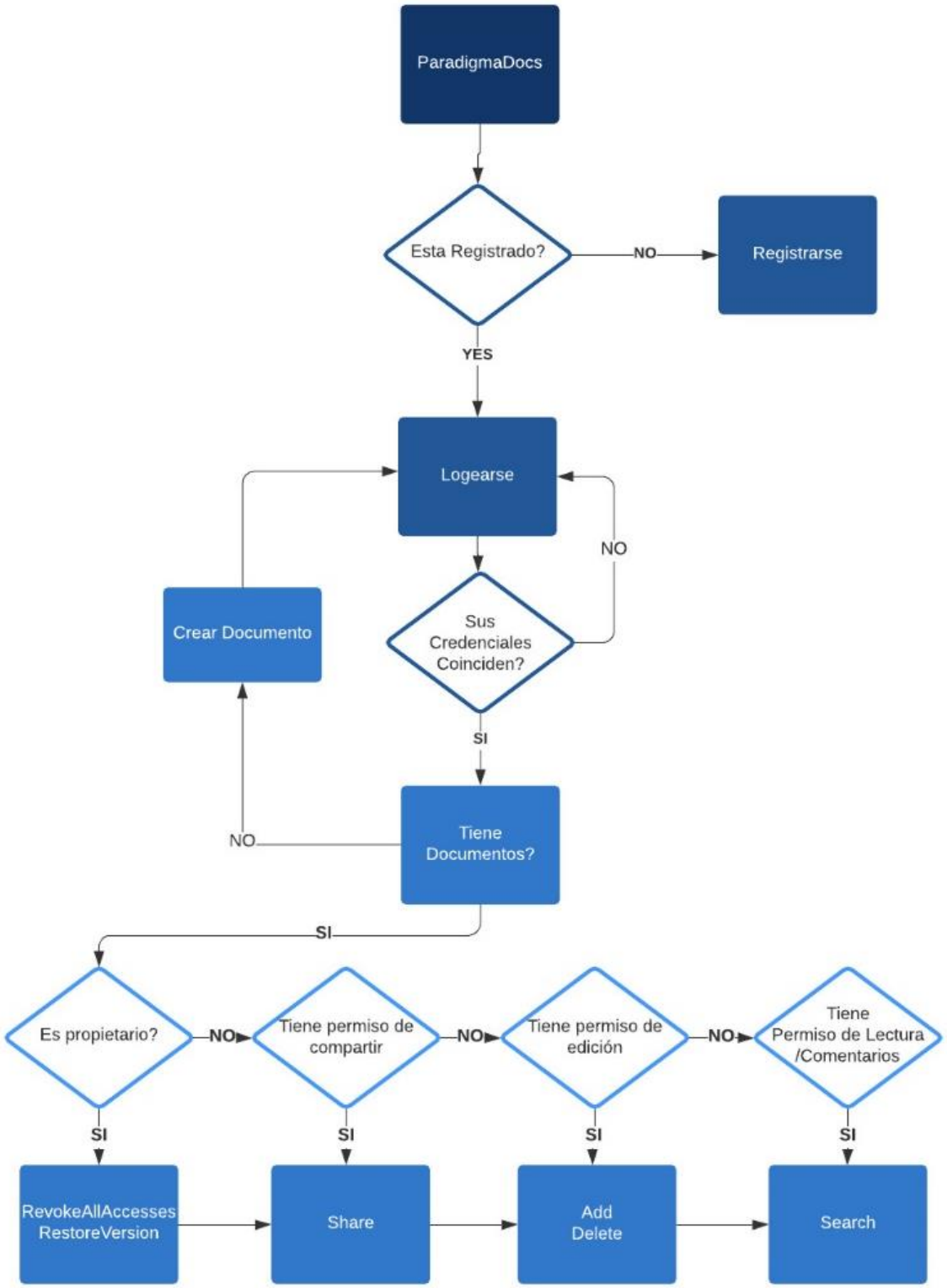
https://www.swiprolog.org/pldoc/doc_for?object=manual

Prolog (lenguaje de programación) - EcuRed. (2022). Retrieved January 2, 2022, from

Ecured.cu website:

https://www.ecured.cu/Prolog_(lenguaje_de_programaci%C3%B3n)

Anexo


ParadigmaDocs
Esta Registrado?
NO
Registrarse
YES
Logearse
NO
Crear Documento
Sus Credenciales Coinciden?
SI
NO
Tiene Documentos?
SI
Es propietario?
NO
Tiene permiso de compartir
NO
Tiene permiso de edición
NO
Tiene Permiso de Lectura /Comentarios
SI
SI
SI
SI
RevokeAllAccesses
RestoreVersion
Share
Add
Delete
Search

Anexo 1: Diagrama de jerarquía de accesos

Anexos Ejemplos (Se muestran en formato paradigmaDocsToString) para facilitar su lectura

```
> > > > > 2DocKim2 id 3 < < < < <
* Creado por kim
        -> Usuarios Con Accesso
        -> Historial
                ------> Version Activa <------
                * * * * * Version 0 * * * * *
                Creada el 20 de Diciembre 2020
                2ContenidoKim2
```

Anexo 2: Ejemplo Create

```
> > > > > DocKim id 2 < < < < <
* Creado por kim
       -> Usuarios Con Accesso
          * chuck Permiso de Escritura, Lectura, Compartir
          * saul Permiso de Escritura, Lectura, Compartir
       -> Historial
```

Anexo 3: Ejemplo Share

```
> > > > > DocKim id 2 < < < < <
* Creado por kim
       -> Usuarios Con Accesso
          * chuck Permiso de Escritura, Lectura, Compartir
          * saul Permiso de Escritura, Lectura, Compartir
       -> Historial
```

Anexo 4: Ejemplo Revoke All Accesses

```
------> Version Activa <------
* * * * * Version 2 * * * * *
Creada el 12 de Agosto 2021
ContenidoKim Contenido Aniadido por Chuck/Contenido Aniadido por Kim
* * * * * Version 1 * * * * *
Creada el 12 de Agosto 2021
ContenidoKim Contenido Aniadido por Chuck
* * * * * Version 0 * * * * *
Creada el 20 de Diciembre 2020
ContenidoKim
```

Anexo 5: Ejemplo Add

```
-> Historial
       ------> Version Activa <------
       * * * * * Version 0 * * * * *
       Creada el 20 de Diciembre 2020
       ContenidoKim
       * * * * * Version 2 * * * * *
       Creada el 12 de Agosto 2021
       ContenidoKim Contenido Aniadido por Chuck/Contenido Aniadido por Kim
       * * * * * Version 1 * * * * *
       Creada el 12 de Agosto 2021
       ContenidoKim Contenido Aniadido por Chuck
```

Anexo 6: Ejemplo Restore Version

```
------> Version Activa <------
* * * * * Version 2 * * * * *
Creada el 12 de Agosto 2021
ContenidoSaul/Contenido Aniadid
* * * * * Version 1 * * * * *
Creada el 12 de Agosto 2021
ContenidoSaul/Contenido Aniadido por Saul
```

Anexo 7: Ejemplo Delete

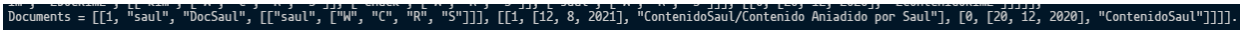
```
* * * * * * * Google docs * * * * * * *
Creado el 20 de Diciembre 2020

******** Usuarios Registrados ********
Usuario: chuck
Password: qwerty
Registrado el 20 de Diciembre 2020

Usuario: saul
Password: 1234
Registrado el 29 de Octubre 2021

Usuario: kim
Password: 4321
Registrado el 12 de Agosto 2021

************* Documentos ************
> > > > > DocChuck id 0 < < < < <
* Creado por chuck
        -> Usuarios Con Accesso
        -> Historial
                ------> Version Activa <------
                * * * * * Version 0 * * * * *
                Creada el 20 de Diciembre 2020
                Contenidochuck

> > > > > 2DocKim2 id 3 < < < < <
* Creado por kim
        -> Usuarios Con Accesso
        -> Historial
                ------> Version Activa <------
                * * * * * Version 0 * * * * *
                Creada el 20 de Diciembre 2020
                2ContenidoKim2
```

Anexo 8.1: Ejemplo ParadigmaDocsToString Versión todos los documentos

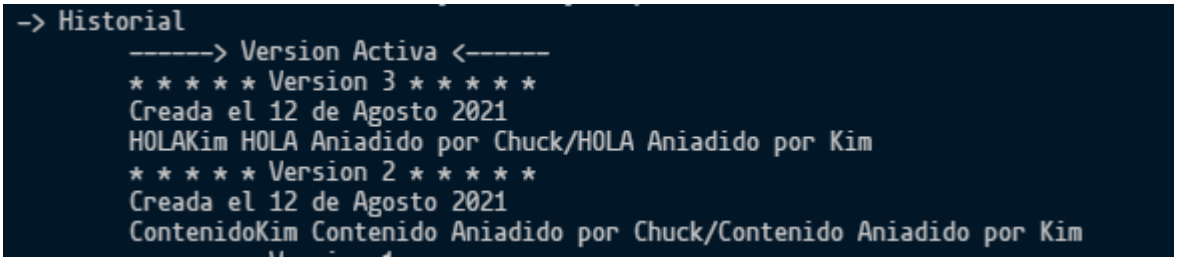
```
* * * * * * * * *  kim  * * * * * * * * *

********** Es Propietario de **********
> > > > > 2DocKim2 id 3 < < < < <
* Creado por kim
        -> Usuarios Con Accesso
        -> Historial
                ------> Version Activa <------
                * * * * * Version 0 * * * * *
                Creada el 20 de Diciembre 2020
                2ContenidoKim2

> > > > > DocKim id 2 < < < < <
* Creado por kim
        -> Usuarios Con Accesso
           * chuck Permiso de Escritura, Lectura, Compartir
           * saul Permiso de Escritura, Lectura, Compartir
        -> Historial
                ------> Version Activa <------
                * * * * * Version 2 * * * * *
                Creada el 12 de Agosto 2021
                ContenidoKim Contenido Aniadido por Chuck/Contenido Aniadido por Kim
                * * * * * Version 1 * * * * *
                Creada el 12 de Agosto 2021
                ContenidoKim Contenido Aniadido por Chuck
                * * * * * Version 0 * * * * *
                Creada el 20 de Diciembre 2020
                ContenidoKim

************ Tiene acceso a *************

> > > > > DocSaul id 1 < < < < <
* Creado por saul
        -> Usuarios Con Accesso
```

Anexo 8.1: Ejemplo ParadigmaDocsToString Versión documentos usuario "kim"

Anexo 9.1: Ejemplo Seach, se busca la palabra "Kim"



Anexo 9.2: Ejemplo Search, se busca la palabra "Saul"



Anexo 10: Ejemplos Search And Replace (Se busca Contenido y se reemplaza por HOLA)