Universidad de Santiago de Chile

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

Informe 1 Paradigma Funcional

Docente: Roberto González Ibáñez

Estudiante: Nícolas Farfán Cheneaux

Diciembre 2021

Contenido

Introducción

Durante este primer bloque de la asignatura vimos los alcances del paradigma funcional, para entender el paradigma en profundidad, se asignó un proyecto de laboratorio estilo ***Google Docs***. el cual se aplican casi es su totalidad todos los conceptos teóricos vistos. El presente informe tiene como objetivo exponer todo el proceso empezando por la descripción del problema, análisis del problema ,diseño de la solución y finalmente evaluar la aplicación del lenguaje de programación y paradigma, limitaciones y ventajas; resultado final del proyecto.

Descripción del problema

Las plataformas de documentos colaborativas tienen múltiples usos y aplicaciones actualmente, sus funciones principales abarcan desde lo simple, crear y escribir un documento propio a lo más complejo editar un documento a las preferencias del usuario, guardarlo en un historial y compartirlo con otros usuarios en línea y además otorgarle permisos sobre el documento, la plataforma de documentos colaborativa en línea más usada actualmente es ***Google Docs***, a lo largo del laboratorio de esta asignatura, se creará una versión propia de esta plataforma, con diferentes lenguajes y paradigmas, particularmente este informe pertenece al primer lenguaje/paradigma a tocar, el lenguaje Scheme/ Racket que funciona bajo el paradigma de programación funcional

Descripción y uso del paradigma

El paradigma funcional se caracteriza por estar orientado a preguntar ¿Qué quieres resolver? Antes que ¿Cómo resolverlo?, en torno a este concepto el código se estructura de funciones las cuales siempre devuelven el mismo valor, es decir es inmutable, esto se puede ver en funciones como filter, map, las cuales devuelven una nueva lista o resultado a la original, junto a esto el concepto de definir una variable no se utiliza en proyecto.

Dentro del paradigma funcional existe una forma ‘pura’ la cual es declarativa al 100%, es decir no es posible usar condicionales dentro del ***main*** o función principal, cada sentencia condicional debe ser encapsulada, para mantener la legibilidad y lógica de este paradigma, el propósito de esto es tener un código entendible por los programadores, ya que los nombres de funciones son creados para entender que hacen estas, por lo tanto una de las ventajas de este paradigma y estilo declarativa es poder mantener el código a largo plazo e identificar la lógica que estamos siguiendo, solo está la intención del código.

Otra ventaja de este paradigma es que todo es reutilizable, las funciones de los TDA fueron creados para reutilizarlos, cada función de cada TDA se usará hasta finalizar el proyecto, estilo piezas de lego, todo lo pequeño se junta para tener la solución final e incluso para añadir nuevas funcionalidades y cambias la representación de esta. El presente proyecto lo demuestra, necesitando funciones que se usaron en la función n°2 en la función n°8. Otro aspecto como se mencionó al inicio la inmutabilidad de las funciones y la no creación de variables son una ventaja pues se evita errores en modificar elementos y se forma una cadena de errores, en cambio al aplicar funciones seguidas se puede identificar cual es la que está fallando.

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.

Siendo estos escalables, es decir cumpliendo una jerarquía: Escritura > Comentarios > Lectura.

Los documentos tienen una estructura particular la cual se rige por lo requerido:
título, contenido (solo texto), id, creador, historial, lista de colaboradores

Sobre esta estructura se tiene el control sobre el documento, el creador tiene control absoluto sobre este, pudiendo realizar las siguientes operaciones:

Símbolos: w: escritura, c: comentarios r: lectura A:autor

- Crear Documento (Cualquier usuario registrado)
- Añadir Texto (A, w)
- Otorgar Permisos (A)
- Eliminar Permisos (A)
- Restaurar una versión (A)
- Eliminar Texto (A, w)
- Comentar (A, w, c)
- Aplicar estilos (A, w)
- Reemplazar texto (A, w)
- Buscar texto (A, w, r, c)
- Aplicar estilos (A, w)
- CtrlZ, CtrY (A, w)

Dentro de los documentos existe una estructura adicional: El historial: Su función es registrar:

- El cambio realizado.
- Que usuario lo realizó.
- La fecha en que se hizo.
- En qué orden se hizo (id).

Datos de salida:

En términos de esta plataforma, el usuario debe logearse para ejecutar una 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

Cómo se puede apreciar en el diagrama *(ver anexo 1),* el usuario previamente a ingresar a la plataforma debe estar registrado, si no, debe hacerlo, para esto, se implementan las funciones de registro y logeo de credenciales.

Operaciones de Registro / Logeo / Permisos de usuarios

Se crea una función **Register** la cual permite al usuario crear una cuenta en la plataforma, esta función registra un **usuario, contraseña, fecha de creación de cuenta,** para que esta función funcione, valga la redundancia, con varios usuarios se debe aplicar en una composición de funciones *(ver anexo2),* ya que cada vez que se aplica retorna una versión actualizada de paradigmadocs.

Cada **Cuenta** tiene la siguiente forma:
(username *password* date)

Por lo tanto en paradigmadocs los usuarios registrados se ven de la siguiente forma: *(ver anexo3)* **((username1 password1 date1) (username2 password2 date2) …etc.)**

Para evitar repeticiones de usuarios se optó por una función *recursiva* que verifique si el **username** es idéntico a uno ya creado anteriormente.

Ya el usuario registrado, para que este pueda realizar cualquier operación sobre los documentos, se requiere una función **Login,** la cual es otra lista *(ver anexo4),* donde solamente se almacena el **username** del único usuario logeado en tal momento, para que esto pueda ocurrir, el **username y *password*** deben estar correctas, es decir deben estar en la lista de los usuarios registrados.

Operaciones sobre los documentos

Antes de hacer cualesquiera operaciones de edición sobre los documentos, el usuario debe tener documentos, la función **create** se encarga de esto, para ello, toma la tercera lista de la plataforma **paradigmadocs** y añade una lista llamada **documento,** la cual consta de la siguiente estructura:

(id autor contenido historial usuarios_con_acceso) *(ver anexo5)*

El primer elemento **id** es el elemento más importante, ya que es único en la lista de documentos, es decir no pueden existir 2 iguales, con este mecanismo se permite a un mismo usuario crear más de 1 documento.
Al momento de crear un documento, este se almacena automáticamente en el **historial** con versión **0** y la fecha de creación es en el cual se creó el documento.

Por lo tanto el historial queda de la siguiente forma:

(*vn… v2* **(id date contenido)** *v1* **(id date contenido))**

Donde v1 tiene id:0 y date es la fecha de creación del documento

Otro elemento importante es la lista de usuarios con los que se comparte el documento, esta es llenada mediante la función **Share**, una accesslist que agrega(usuario X permiso), esto lo veremos más a detalle en la función **Share**. *(ver anexo6)*

Funciones de edición

Dada la representación escogida es necesario estar todo el tiempo, añadiendo, eliminando, actualizando las listas que conforman paradigmadocs, para esto es necesario recorrer de forma recursiva cada una de estas, pero gracias a las funciones propias de el paradigma funcional como filter y map, es posible eliminar o saltar elementos fácilmente. En repetidas ocasiones la cantidad de código usado en una función recursiva puede ser comprimido en tal solo 1 línea al aplicar filter o map.

Ahora que podemos realizar cualquier cambio y retornar una versión actualizada de paradigmadocs, detallamos el proceso que seguirán estas operaciones:

Add *(ver anexo7)*

1) Recurre a la versión activa del documento y añade el texto a esta, mediante **string-join.**
2) Se actualiza paradigmadocs creando una nueva versión en el historial

Restore versión *(ver anexo8)*

Como en nuestra representación la versión activa es aquella que se se encuentra al inicio de la lista de versiones, el proceso a seguir es.

1) Mediante filter, se *salta* la versión que se quiere restaurar, obtenido un historial sin esta versión.
2) Se añade mediante **append** la versión que se quiere restaurar al inicio
3) Se actualiza el documento y se añade a paradigmadocs.

Revoke All Accesses *(ver anexo9)*

1) Mediante filter se busca los documentos a los que el usuario logeado es propietario.
2) Una vez filtrado, mediante **map** se eliminan los permisos y se deja una lista vacía.
3) Se actualiza el documento y se añade a paradigmadocs.

Paradigmadocs –> String *(ver anexo11)*

1) Para obtener todos los datos, se utiliza las funciones declarativas **map y filter**, ya que es necesario extraer información específica de cada documento.
2) Como el resultado anterior son listas, se transforma a string mediante funciones de concatenación de strings.

Delete *(ver anexo12)*

1) Recurre a la versión activa del documento y aplica un corte mediante el manejo de strings, en este caso se usó string-append para el largo del texto y substring para corta el string.
2) Se actualiza el documento y se añade a paradigmadocs.

Search and Replace *(ver anexo13)*

1) Recurre a la versión activa del documento y busca si el texto buscado coincide con el texto de esta versión
2) Si coincide, reemplaza el texto buscado por el ingresado, sino no hace ningún cambio
3) Se actualiza el documento y se añade a paradigmadocs.

ApplyStyles *(ver anexo14)*

1) Recurre a la versión activa del documento y verifica que el texto al que se quiere aplicar estilos exista.
2) Si es así, mediante función de concatenación agrega a ambos lados de el texto los estilos ingresados, solo pueden ser del tipo *#b #i #u* si es ingresado cualquier otro estilo no es aplicado, si el texto buscado no se encuentra, no se aplica ningún cambio sobre el documento.
3) Se actualiza el documento y se añade a paradigmadocs.

Comment *(ver anexo15)*

1) Recurre a la versión activa del documento y verifica que el texto al que se quiere aplicar estilos exista.
2) Si es así añade un comentario mediante funciones de concatenación de strings, obteniendo una versión comentada del documento, si el texto no se encuentra, no se aplica ningún cambio sobre el documento.
3) Se actualiza el documento y se añade a paradigmadocs.

CtrlZ y CtrlY *(ver anexo16.1, anexo 16.2)*

1) Para ambas funciones es necesario aplicar un cambio en la estructura de nuestra plataforma, se crea una memoria ubicada en la lista de documentos, la cual almacena el texto original donde se empezó a hacer CtrlZ y el número de CtrlZ's realizados
2) Funciona de la siguiente manera, al hacer CtrlZ, se aleja de la versión original 0, sumando a esta, al hacer CtrlY se acerca a la versión, restando a esta.
3) Para que CtrlY funcione, se comprueba que anteriormente se hizo CtrlZ, si es así, se procede a descontar de la "memoria" x número de CtrlY,
4) Finalmente se actualiza el documento y se añade a paradigmadocs.

Funciones de búsqueda

Search *(ver anexo10)*

1) Usando filter se obtiene los id de los documentos donde hay **ocurrencias de texto** y el usuario tiene cualquier acceso.
2) Sobre la lista resultante de filter, se aplica map para dar como resultado final, todos los documentos donde se tiene ocurrencia de texto.

Funciones de encriptación *(ver anexo17)*

El método consiste en transformar una contraseña en caracteres y estos caracteres tienen una representación numérica **integer** la cual puede ser modificada independiente de que letra sea, ejemplo: abc = 123 , a cada número se le hace cualquier operación aritmética (* 2) = 246, transformando a carácter: bdf, tenemos un texto encriptado, finalmente para desencriptar se realiza el mismo proceso pero con la operación aritmética inversa a la realizada, en este caso (/ 2), resultando el texto original : abc.

Consideraciones de Implementación

Estructura del proyecto

El proyecto se estructura de main y 7 TDA's para cubrir todas las características requeridas, a continuación se describe las funciones de los TDA usados.

TDA Paradigmadocs: En encargado de crear y modificar la estructura principal de paradigamdocs y cualquier lista de esta (Register,Logeados, Documentos).

TDA Fecha: Encargado de proveer la estructura básica de la fecha cuando sea requerida, maneja excepciones y comprobaciones para que la fecha sea correcta.

TDA User: Encargado de las operaciones básicas antes de hacer cualquier modificación a los documentos, como por ejemplo registrar al usuario, logear al usuario, deslogearlo.

TDA Documento: Encargado de las operaciones básicas sobre los documentos y su representación, este TDA necesita de 2 TDA extras para funcionar, el TDA Access y el TDA Versión.

- **TDA Access:** Encargado de manejar los permisos que se pueden otorgar sobre un documento, además de registrar las AccessList que se apliquen sobre paradigmadocs, como styles o Access, además filtra permisos y obtiene el tipo de usuario que es en paradigmadocs (editor, comentador, lector, etc).
- **TDA Versión:** Encargado de manejar las versiones o historial de un documento, obtener la versión activa, filtrar versiones con una característica en específico, etc.
- **TDA Memoria:** Únicamente usado por las funciones CtrlZ y CtrlY, provee las operaciones básicas que se pueden realizar sobre la memoria del documento, almacenar y hacer cambios sobre esta.

Bibliotecas empleadas

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

Compilador o Interprete usado

Dr. Racket 8.2

Razones de la elección

Los TDA son necesarios para mantener un orden en el código y hacerlo mantenible a largo plazo, además de estructurar los componentes básicos de una plataforma de documentos, el intérprete fue tomado por defecto, no se consideró otra versión, y no se vio en la necesidad de usar alguna biblioteca.

Instrucciones de uso, ejemplos de entrada

Constructor paradigmadocs:

(define emptyGDocs (paradigmadocs "NICO DOCS" (date 25 10 2021) encryptFn decryptFn))
- Register:
(define gDocs1 (register (register (register emptyGDocs (date 25 10 2021) "user1" "pass1") (date 26 10 2021) "user2" "pass2") (date 27 10 2021) "user3" "pass3"))
- Create:
(define gDocs2 ((login gDocs1 "user1" "pass1" create) (date 30 08 2021) "doc1" "Documento1 creado por user1"))
- Share:
(define gDocs003 ((login gDocs03 "user2" "pass2" share) 1 (access "user1" #\i)(access "user2" #\c)(access "user3" #\w)(access "user3" #\q)))
- Add:
(define gDocs4 ((login gDocs00003 "user2" "pass2" add) 0 (date 20 10 2021) "Add"))
- Restore Version:
(define gDocs5 ((login gDocs000004 "user1" "pass1" restoreVersion) 0 1))
- Revoke All Accesses:
(define gDocs6 (login gDocs0005 "user1" "pass1" revokeAllAccesses))
- Search:
((login gDocs6 "user1" "pass1" search) "por")
- Paradigmadocs->String
(login gDocs6 "user3" "pass3" paradigmadocs->string)
- Delete
(define gDocs7 ((login gDocs6 "user1" "pass1" delete) 0 (date 30 11 2021) 0))
- Search and replace
(define gDocs8 ((login gDocs7 "user1" "pass1" searchAndReplace) 0 (date 20 10 2021) "po" "CHANGE"))
- Apply Styles
(define gDocs9 ((login gDocs8 "user1" "pass1" applyStyles) 0 (date 20 10 2021) "Documento1 creado" #\b #\u #\i #\y))
Comment
(define gDocs10 ((login gDocs9 "user2" "pass2" comment) 2 (date 20 10 2021) "1" "comentario"))
- Encrypt Decrypt
(decryptDn (encryptDn "contraseña1234"))
- CtrlZ& CtrlY
(define gDocs00011 ((login ((login gDocs10 "user1" "pass1" ctrlZ)0 3) "user1" "pass1" ctrlY)0 3))

Resultados Esperados

Se espera que el proyecto funcione correctamente con todas las funciones sin importar el orden en que se ingresan, también se espera que todas las posibles excepciones que puedan ocurrir estén correctamente administradas y finalmente se espera que el código sea mantenible en el tiempo, de forma que pueda ser modificado sin problemas.

Posibles Errores

Es posible que exista una combinación de operaciones no compatibles o que no se haya considerado una excepción al ejecutar las “operation” en orden distinto al que se muestran los ejemplos, también es posible que existan funciones redundantes o que no son necesarias para una tarea en específico.

Resultados y Autoevaluación

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

Evaluación

Requerimiento	Tipos de pruebas realizadas	Exitosas	Fracasos	Razón de fallo	Se completó?
TDA	Usar los selectores con las otras funciones				
Register	Realizar la función con recursividad natural y de cola	Al usar ambos tipos de recursividad.	Usando solo recursividad de cola.	No se cumplen los requerimientos obligatorios	
Login	Realizar la función con recursividad natural y de cola, realizar una currificación no correcta	Se cambia la representación a una semicurrificaicón en cada operation	Usar cond para currificar el login directamente.	No se respeta que la función Login es agnóstica a la operation realizada	
Create	Se usa la función sin logeo y un usuario no registrado	Se usa una función que verifique el usuario está logeado	El usuario no está logeado	No se consideró tal excepción	
Share	Con usuarios no registrados, con permisos no válidos	Se debe considerar una función que filter permisos válidos	Cuando no se usa la función para filtrar permisos	No manejar correctamente las excepciones	
Add	Un usuario sin permisos de edición usa la función y no tiene documentos	Se usa una función que verifique que el usuario es editor	El usuario no tiene acceso de editor a documentos	No se consideró tal excepción	
Restore Version	Restaurar una versión que no existe, se usa la función sin usar login.	Se usa una función para verificar que el usuario es propietario, manejar la excepción cuando el id no existe	No verificar si es propietario de tal documento	No haber considerado tal excepción	
Revoke All Accesses	Se usa la función sin logeo y un usuario no registrado, se usa la función sin usar login. Se usa map	Se usa una función para verificar que el usuario es propietario.			
Search	Se usa la función sin logeo y un usuario no registrado, se usa filter, el usuario no tiene acceso a ningún documento.	Se considera una función que verifique las ocurrencias en los documentos que tiene acceso.			
Paradigmadocs->String	Se usa la función usar login y usando login	Juntar ambas condiciones en la misma función y verificar si el usuario está logeado.			

Delete	Se usa la función sin logeo, el propietario no tiene permiso de edición, el texto a borrar es mayor al largo del original	Considerar la excepción dentro de la función			
Search and Replace	Buscar “”, se usa la función sin logeo, el usuario no tiene documentos ni permisos de escritura	Considerar la excepción dentro de la función y usar la función para ver si el usuario es editor			
Apply Styles	Estilos no permitidos, aplicar estilos a un texto vacío, usar la función sin login	Usar la función para ver si el usuario es editor, se filtra los estilos permitidos			
Comment	Comentar un documento donde la versión activa está comentada	Crear una función que retorna la versión activa sin comentarios para comentarla	Conflicto con las funciones Search al buscar símbolos y caracteres de comment		
Encriptación	Texto normal	No existe ninguna excepción al ser una función agnóstica			
CtrlZ y CtrlY	Usar CtrlY antes que CtrlZ	Antes que nada se debe considerar una memoria al usar por primera vez CtrlZ, entonces una función es la encargada de comprobar que exista memoria (Se hizo CtrlZ antes)		Pueden existir excepciones en su funcionamiento	

Las operaciones CtrlZ y CtrY funcionan correctamente con los ejemplos dados pero pueden existir excepciones no consideradas para su correcto funcionamiento y es posible que no funcione el 100% de las veces.

Conclusión

Alcances

En la realización de este proyecto se ha usado la esencia del paradigma funcional, solo usando recursividad, sin variables y usando funciones declarativas, además el uso de los TDA para representar abstracciones de los elementos más importantes de la plataforma de documentos es una gran ventaja para la lectura y entendimiento a largo plazo del código, al contrario de usar solamente las funciones nativas del lenguaje como *car, cdr, list, append*, sería confuso saber de qué lista o función proviene tal parámetro o operación.

Limitaciones

La principal limitación es la falta de experiencia en paradigma funcional, la poca práctica en recursividad y la falta de costumbre al leer documentación de un lenguaje de programación.

Dificultades de usar el paradigma para abordar el problema

Personalmente el uso de un nuevo lenguaje y paradigma en un tiempo relativamente corto fue un gran desafío, por lo tanto fue necesario dedicarle muchas horas, paciencia para entender todos los conceptos correctamente para luego construir un proyecto con funciones útiles.

Durante el camino se presentó un gran problema y es no haber examinado el proyecto correctamente y haber entendido la estructura de la plataforma, había realizado hasta la función *delete* con una notación incorrecta y no usando correctamente los TDA con muchos constructores propios de Racket más no de los TDA definidos por lo tanto procedí a borrar todo realizarlo nuevamente, desde cero. Afortunadamente quedaba mucho tiempo disponible y al realizarlo nuevamente realmente pude apreciar la ventaja de haber definido previamente un TDA pues aunque sea difícil al inicio de entender, los TDA's están para tener un buen uso de la abstracción de los elementos que componen un elemento, facilitan la forma en la que se lee el código y se mantiene en el tiempo.-

Referencias

- Dybvig, R. K. The Scheme programming language. Mit Press, 2009, https://www.scheme.com/tspl4

-*Racket Documentation*. (n.d.). Racket Documentation

https://docs.racket-lang.org/

-Wikipedia contributors. (2021, November 10). *Functional programming*. Wikipedia.

https://en.wikipedia.org/wiki/Functional_programming

-*Why am I falling in love with functional programming?* (n.d.). Morioh. Retrieved June 10,

2022, from https://morioh.com/p/17ba816a27db

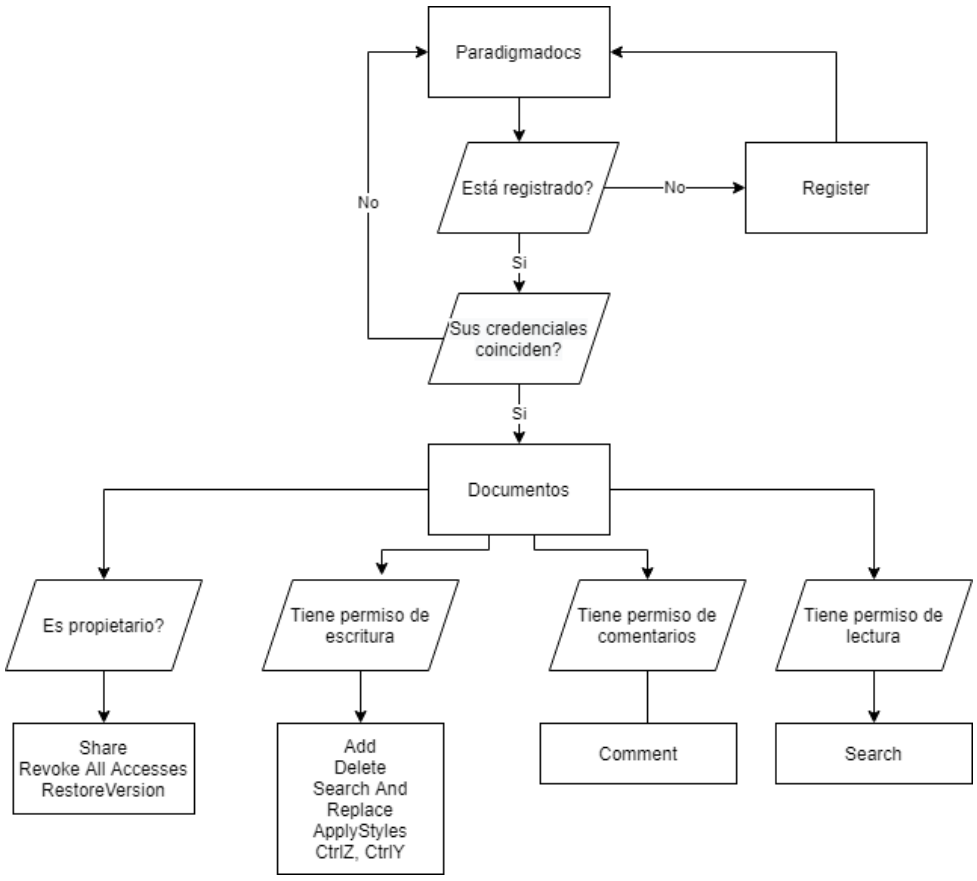
-*Gonzales, R. (2021). "Proyecto Semestral de Laboratorio". Enunciado de Proyecto*

Paradigmas de Programación from

https://docs.google.com/document/d/1pORdJwp5WJ7V3DIAQik9B58llpdxpurQRVTKPZH

OpS8/edit

Anexo



Anexo1: *Diagrama jerárquico de permisos para usar operations*

(define gDocs1 (register (register (register emptyGDocs (date 25 10 2021) "user1" "pass1") (date 26 10 2021) "user2" "pass2") (date 27 10 2021) "user3" "pass3"))

Anexo 2: *Ejemplo de registro*

```
'("NICO DOCS"
  (25 10 2021)
  #<procedure:encryptFn>
  #<procedure:decryptFn>
  (("user1" "1ssap" (25 10 2021)) ("user2" "2ssap" (26 10 2021)) ("user3" "3ssap" (27 10 2021)))
  ()
  ())
```

Anexo 3: *3 usuarios registrados en paradigmadocs*

```
'("NICO DOCS"
  (25 10 2021)
  #<procedure:encryptFn>
  #<procedure:decryptFn>
  (("user1" "1ssap" (25 10 2021)) ("user2" "2ssap" (26 10 2021)) ("user3" "3ssap" (27 10 2021)))
  ()
  (("doc1" "user1" 0 ((0 (30 8 2021) "1resu rop odaerc 1otnemucoD")) ()))))
```

Anexo 4: *2da lista de paradigmadocs usuarios logeados, la lista se encuentra vacía pues al realizar una operation al final el usuario se deslogea*

```
(("doc4" "user1" 3 ((0 (30 10 2021) "1resu rop odaerc 2otnemucoD")) ())
 ("doc3" "user3" 2 ((0 (30 10 2021) "3resu rop odaerc 1otnemucoD")) ())
 ("doc2" "user2" 1 ((0 (30 9 2021) "2resu rop odaerc 1otnemucoD")) ())
 ("doc1" "user1" 0 ((0 (30 8 2021) "1resu rop odaerc 1otnemucoD")) ()))
```

Anexo 5: *Se crean 4 documentos con id's únicas*

```
(("doc4" "user1" 3 ((0 (30 10 2021) "1resu rop odaerc 2otnemucoD")) (("user3" #\w) ("user2" #\c)))
 ("doc3" "user3" 2 ((0 (30 10 2021) "3resu rop odaerc 1otnemucoD")) (("user2" #\c) ("user1" #\w)))
 ("doc2" "user2" 1 ((0 (30 9 2021) "2resu rop odaerc 1otnemucoD")) (("user3" #\w)))
 ("doc1" "user1" 0 ((0 (30 8 2021) "1resu rop odaerc 1otnemucoD")) (("user2" #\w) ("user3" #\r))))
```

Anexo 6: *Se añaden permisos sobre los documentos*

```
("doc3"
 "user3"
 2
 ((2 (21 10 2021) "3cod ne odañA 2droW 3resu rop odaerc 1otnemucoD")
  (1 (21 10 2021) "2droW 3resu rop odaerc 1otnemucoD")
  (0 (30 10 2021) "3resu rop odaerc 1otnemucoD"))
 (("user2" #\c) ("user1" #\w)))
```

Anexo 7: *Se añade texto sobre el documento 3 (id 2)*

```
(("doc3"
  "user3"
  2
  ((0 (30 10 2021) "3resu rop odaerc 1otnemucoD")
   (2 (21 10 2021) "3cod ne odañA 2droW 3resu rop odaerc 1otnemucoD")
   (1 (21 10 2021) "2droW 3resu rop odaerc 1otnemucoD"))
  (("user2" #\c) ("user1" #\w)))
```

Anexo 8: *Se restaura la versión 0 del documento 3 (id 2), versión activa: 0*

```
("doc3"
 "user3"
 2
 ((0 (30 10 2021) "3resu rop odaerc 1otnemucoD")
  (2 (21 10 2021) "3cod ne odañA 2droW 3resu rop odaerc 1otnemucoD")
  (1 (21 10 2021) "2droW 3resu rop odaerc 1otnemucoD"))
 ())))
```

Anexo 9: *Se eliminan todos los accesos sobre el documento 3 (user2, user1)*

```
> ((login gDocs6 "user2" "pass2" search) "Doc")
'(("doc2" "user2" 1 ((1 (21 10 2021) "2cod odañA oN 2resu rop odaerc 1otnemucoD") (0 (30 9 2021) "2resu rop odaerc 1otnemucoD")) (("user3" #\w)))
  ("doc3"
   "user3"
   2
   ((0 (30 10 2021) "3resu rop odaerc 1otnemucoD") (2 (21 10 2021) "3cod ne odañA 2droW 3resu rop odaerc 1otnemucoD") (1 (21 10 2021) "2droW 3resu rop odaerc
1otnemucoD"))
   (("user2" #\c) ("user1" #\w))))
```

Anexo 10: *Se busca el texto "Doc" en los documentos que user2 tiene acceso*

```
********** NICO DOCS **********
Creado el 25 de Octubre de 2021

➢USUARIOS REGISTRADOS:
Usuario: user1
Contraseña: 1ssap
Registrado el 25 de Octubre de 2021
-----------------------------------
Usuario: user2
Contraseña: 2ssap
Registrado el 26 de Octubre de 2021
-----------------------------------
Usuario: user3
Contraseña: 3ssap
Registrado el 27 de Octubre de 2021
-----------------------------------
```

```
➢DOCUMENTOS:
doc3
Creado el 30 de Octubre de 2021
El propietario es user3
id: 2
Historial de Documentos:
        * * Versión Activa * *
        Versión nº 0
        Ultima Modificación el 30 de Octubre de 2021
        Contenido: Documento1 creado por user3
        * * * * * * * * * * *
        Versión nº 2
        Ultima Modificación el 21 de Octubre de 2021
        Contenido: Documento1 creado por user3 Word2 Añado en doc3
        * * * * * * * * * * *
        Versión nº 1
        Ultima Modificación el 21 de Octubre de 2021
        Contenido: Documento1 creado por user3 Word2
        * * * * * * * * * * *
Usuarios con acceso:
user2 → permiso de comentarios
user1 → permiso de escritura
```

Anexo11: *Ejemplo corto de paradigmadocs->string*

```
("doc1"
 "user1"
 0
 ((3 (30 11 2021) "rop odaerc 1otnemucoD")
  (1 (20 10 2021) "ddA 1resu rop odaerc 1otnemucoD")
  (2 (21 10 2021) "arbalaP ddA 1resu rop odaerc 1otnemucoD")
  (0 (30 8 2021) "1resu rop odaerc 1otnemucoD"))
 ())))
```

Anexo 12: *Se eliminan los 10 últimos caracteres del contenido de la versión antigua, creando una nueva versión*

```
("doc1"
 "user1"
 0
 ((3 (20 10 2021) "ddA 1resu rEGNAHC odaerc 1otnemucoD")
  (1 (20 10 2021) "ddA 1resu rop odaerc 1otnemucoD")
  (2 (21 10 2021) "arbalaP ddA 1resu rop odaerc 1otnemucoD")
  (0 (30 8 2021) "1resu rop odaerc 1otnemucoD"))
 ())))
```

Anexo 13: *Se reemplaza la palabra "po" por CHANGE, creando una nueva versión*

```
("doc1"
 "user1"
 0
 ((4 (20 10 2021) "ddA 1resu rEGNAHC  i\\# u\\# b\\# odaerc 1otnemucoD i\\# u\\# b\\# ")
  (3 (20 10 2021) "ddA 1resu rEGNAHC odaerc 1otnemucoD")
  (1 (20 10 2021) "ddA 1resu rop odaerc 1otnemucoD")
  (2 (21 10 2021) "arbalaP ddA 1resu rop odaerc 1otnemucoD")
  (0 (30 8 2021) "1resu rop odaerc 1otnemucoD"))
 ())))
```

Anexo 14: *Se añaden estilos a la palabra "Documento1 creado", creando nueva versión*

```
("doc3"
 "user3"
 2
 ((3 (20 10 2021) "3resu rop odaerc  %c]oiratnemoc[c%>-1otnemucoD")
  (0 (30 10 2021) "3resu rop odaerc 1otnemucoD")
  (2 (21 10 2021) "3cod ne odañA 2droW 3resu rop odaerc 1otnemucoD")
  (1 (21 10 2021) "2droW 3resu rop odaerc 1otnemucoD"))
 (("user2" #\c) ("user1" #\w)))))
```

Anexo 15: *Se comenta la palabra " Documento1 " con el comentario "comentario"*

```
(("doc1"
  "user1"
  0
  ((4 (20 10 2021) "ddA 1resu rop odaerc 1otnemucoD" ("ddA 1resu rEGNAHC  i\\# u\\# b\\# odaerc 1otnemucoD i\\# u\\# b\\# " 2))
   (3 (20 10 2021) "ddA 1resu rEGNAHC odaerc 1otnemucoD")
   (1 (20 10 2021) "ddA 1resu rop odaerc 1otnemucoD")
   (2 (21 10 2021) "arbalaP ddA 1resu rop odaerc 1otnemucoD")
   (0 (30 8 2021) "1resu rop odaerc 1otnemucoD"))
  ())
```

Anexo16.1: *Se hace CtrlZ 2 veces, partiendo de la versión 4 a la versión 2*

```
(("doc1"
  "user1"
  0
  ((4 (20 10 2021) "ddA 1resu rEGNAHC odaerc 1otnemucoD" ("ddA 1resu rEGNAHC  i\\# u\\# b\\# odaerc 1otnemucoD i\\# u\\# b\\# " 1))
   (3 (20 10 2021) "ddA 1resu rEGNAHC odaerc 1otnemucoD")
   (1 (20 10 2021) "ddA 1resu rop odaerc 1otnemucoD")
   (2 (21 10 2021) "arbalaP ddA 1resu rop odaerc 1otnemucoD")
   (0 (30 8 2021) "1resu rop odaerc 1otnemucoD"))
  ())
```

Anexo 16.2: *Luego de haber hecho CtrlZ 2, se hace CtrlY 1 restaurando una versión desde donde nos encontramos*

```
> (decryptDn (encryptDn "paradigma_funcional"))
"paradigma_funcional"
```

Anexo 17: Ejemplo de encriptación y desencriptación

Nota: Todos estos ejemplos son extraídos de los ejemplos del archivo 'main'