INFORME PARADIGMAS DE PROGRAMACIÓN

Autor: Nicolás Venegas

Asignatura: Paradigmas de Programación

07 de Diciembre 2021

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

Tabla de Contenidos

1. Introducción

Para este informe se tiene como proyecto la elaboración de un editor de texto colaborativo que siga el estilo de Google Docs, esto mediante el lenguaje Racket(derivado de Scheme) siguiendo una serie de restricciones y requerimientos a la hora de elaborarlo, por lo que más adelante se verá la estructura de resolución, la idea de soluciones y la forma de implementación de las mismas mediante el lenguaje dicho anteriormente seguido finalmente por unos ejemplos de código para poder probar el proyecto elaborado.

2. Descripción del problema

Para este laboratorio se tiene la falta de un editor de texto colaborativo al estilo de Google Docs por lo que se solicitó la creación del mismo usando una tecnología en específico, siendo en este caso Dracket.

3. Descripción breve del paradigma y los conceptos del mismo que se ven aplicados en este proyecto

En este desarrollo se ocupó el paradigma funcional el cual está basado en el uso de verdaderas funciones matemáticas, esto quiere decir que cualquier elemento, proceso, entidad puede ser representado en funciones.

En la construcción del programa se van a diferenciar 2 términos regularmente:

- Recursión: es el proceso donde una función se llama a sí misma con un objetivo, esta puede ser de cola, natural u arbórea dependiendo de cómo se implemente.

- Función Lambda/anónima: es una función que carece de nombre pero que puede ser perfectamente utilizada al declararse.

4. Análisis del problema respecto de los requisitos específicos que deben cubrir

Como se mencionó anteriormente se nos pide programar un editor de texto colaborativo del estilo Google Docs pero este tiene una lista de funcionalidades mínimas para que se considere como exitoso, siendo estas:

- Creación de la plataforma
- Registro de usuarios
- Logeo de usuarios
- Creación de documentos
- Compartir documentos entre usuario
- Agregar contenido a un documentos
- Restaurar Versiones de un documento
- Revocar accesos de un documento
- Buscar documentos
- Mostrar información del usuario junto a la plataforma creada

Como se puede ver en esta lista de requerimientos ya se puede ir deslumbrando lo que se necesitará para cada estructura, por ejemplo ya sabemos que una plataforma requerirá una lista con usuarios y que estos necesitan un nombre y contraseña para poder loguearse.

También es posible visualizar la necesidad de las siguientes estructuras representativas que contengan distintas características::

Estructura/TDA	Componentes
Usuarios	Nombre Contraseña
Versión	Id Fecha Contenido
Acceso	Usuario Tipo Acceso
Documento	Nombre Autor Fecha de creación Lista de usuarios compartidos Contenido
Plataforma	Nombre plataforma Fecha de creación Lista de usuarios registrados Lista de documentos usuario Activo

	Funcion Encriptar Funcion Desencriptar

Por último cada estructura debe tener sus propias funcionalidades internas, que al actuar en conjunto a la de las demás estructuras es posible realizar las tareas pedidas anteriormente.

5. Diseño de la solución

En esta sección de diseño de solución se especificará qué funciones fueron necesarias para los TDA/estructuras con el fin de crear las funcionalidades mínimas.

Antes de todo fue la preparación y programación de los TDA de las estructuras dichas en el análisis de solución a partir de eso se creó un constructor, selector y editor para cada elemento del TDA, las funciones secundarias serían creadas a medida que fueran necesarias para la principal.

- **Creación de la plataforma/definición Paradigmadocs**
    - Para esta función sólo fue necesario programar los constructores, selectores y modificadores de los TDA dichos en el punto anterior.

- **Registro de usuarios**
    - En este caso fue necesario un modificador de la lista de usuarios en un paradigmadocs de tal forma que mediante una función de apoyo que agregue un usuario irrepetible(usando recursión de cola) a una lista, luego solo faltaria “setear” esa nueva lista con el modificador en el paradigmadocs.

- **Logeo de usuarios**
    - funciones necesarias para los TDA:

        - Usuarios/comprobar nombre y password
            - Como indica esta función, a un usuario ya creado se le pasa un nombre y contraseña para ver si ambos coinciden con los del usuario.

        - Paradigmadocs/logear
            - modifica el elemento de usuario activo de un paradigmadocs seteando un usuario pasado como parámetro de manera previa.

    - De manera contigua se creó una función de apoyo, que mediante la recursión natural y la funcionalidad de comprobar nombre y password, revisa cada usuario de la lista en

paradigmadocs viendo si el nombre y contraseña pasados coinciden con el de algún usuario.

    - Finalmente se unió todo de forma similar a esto: “si con el nombre y contraseña pasados es posible logear, devuelveme la el paradigma logeado con el nombre”.

- **Creación de documentos**

    - Para esto fue necesario crear El TDA documento siguiendo la estructura de la tabla, y para el TDA paradigmadocs se creó la función de deslogueo que simplemente desloguea a un usuario de paradigmadocs y una función que agregue un documento ya creado a la lista de documentos.
    - Y toda la función fue planteada de la siguiente manera: “si hay algún usuario logueado crea un documento y añadelo al paradigma y desloguea, en caso contrario no hagas nada ”.

- **Compartir documentos entre usuarios**
    - En esta fue necesario una función que agregue un acceso a un documento, haciendo uso de esa funcion tambien se creo una que agregue una lista de accesos a un paradigma por lo que solo quedó agregar la lista de accesos al documento(que fue buscado por id mediante una función de paradigmadocs) solo si fue ingresado un paradigmadocs donde hubiera un usuario logueado.

- **Agregar contenido a un documentos**
    - En esta fue necesario un función que agregue contenido al final de un texto en un documento, como también una que agregue una versión al documento, luego solo quedó añadir el contenido al documento buscado por id, y a guardar la versión sin agregar el contenido a la lista de versiones, todo por supuesto si hay algún usuario activo logueado.

- **Restaurar Versiones de un documento**
    - Esta fue muy simple ya que se reutilizaron la mayoría de funciones de la funcionalidad anterior, cambiando que en vez de agregar contenido al final del documento este se setea con el contenido guardado en la versión buscada por id (funcion creada en TDA Documento) y el contenido previo fue guardado en una nueva versión.

- **Revocar accesos de un documento**
    - Para esta fue necesario crear una función que logre revocar todos los accesos compartidos en un documento siempre y cuando un usuario pasado coincida con el autor por lo que luego se llamó a esta función con la ayuda de map en toda la lista de

documentos y gracias a una función anónima se seteo el usuario pedido como el usuario logueado, luego la lista resultante fue seteada como la nueva lista de documentos en paradigmadocs, obviamente esto solo ocurre si hay usuario logueado, en caso contrario no hay cambios en la plataforma.

- **Buscar documentos**
    - En este caso se creó una función que verifique si un usuario tiene participación en algún documento (tanto como autor como si está en la lista de compartidos) luego se ocupó esa función como filtro en la lista de documentos usando al usuario logueado y finalmente con la nueva lista de documentos se volvió a filtrar donde el contenido contenga las palabras ingresadas para buscar.

- **Mostrar información del usuario junto a la plataforma creada**
    - en esta se crearon funciones en varios tda para que se muestre información, primero se partió con la construcción de un String que muestre la información de un acceso, luego otro que muestre la información de un documento (todos en sus respectivos Tda) y finalmente otro que muestre la información de un paradigmadocs y gracias a la función map y string-join se puede ir de menos a más, por ejemplo a la hora de mostrar los accesos en la información de documento se llamó a la función map con la función de información de accesos a la lista de los mismos en el documento y a la hora de mostrar la información de la plataforma nuevamente se ocupan las 2 funciones dichas previamente(map y string-join) para poder mostrar la información de todos los documentos sin complicaciones, si se quiere mostrar distinta información solo se tiene que acotar la lista que se llame en map,

6. Aspectos de implementación

Para la elaboración de este proyecto se utilizó el lenguaje Racket, derivado del lenguaje de programación Scheme, con el compilador DRracket en su versión 6.11.

De igual manera no fue necesaria ninguna biblioteca adicional además de la que proporciona el mismo Racket, ya que todas las otras funciones y estructuras necesarias fueron escritas en distintos archivos dependiendo de la necesidad, este método fue elegido ya que permite una mejor estructuración y lectura del código y otorga mayor eficiencia a la hora de modificarlo.

7. Instrucciones de uso

En cuanto a los resultados esperados de la funciones se puede decir que estas funcionan buena manera exceptuando los casos donde se introduzcan datos erróneos o inexistentes, arrojando un error, esto ocurre debido a que este lenguaje y el laboratorio en general está orientado a usuarios que ya saben usar el lenguaje y tienen un plan de lo que quieren hacer por lo que a pesar de que es posible ocuparse de esos errores no nos corresponde hacerlo (en otras palabras no es nuestra responsabilidad).
En la sección de anexos es posible encontrar los ejemplos de todas las funciones siguiendo un camino entendible.

8. Resultados y autoevaluación

Durante la creación del programa y desarrollo del laboratorio surgieron bastantes errores respecto a las funciones mínimas requeridas pero todos estos fueron solucionados dando de esta manera una versión estable y correcta del proyecto, por lo que la autoevaluación me concedió un aprobado ya que todos los requerimientos fueron cumplidos.

9. Conclusiones

Luego de realizar este proyecto es posible ver la programación con un nuevo punto de vista, a pesar de que hubo complicaciones al aplicar este punto de vista también es posible decir que ahora la versatilidad y proponer nuevas ideas o formas de abordar un problema han aumentado.

También es posible decir que para los siguientes laboratorios hacer un cambio de paradigma va a ser más fácil debido a que el primero(este) ya fue realizado.

Para Finalizar decir que siento que las funciones creadas pueden ser mucho más mejoradas, tanto en eficiencia como en eficacia por lo que aún queda por aprender en este punto de vista de la programación funcional.

10. ANEXOS

- Ejemplos
  ; se definen 3 paradigmadocs y se guardan con sus respectivos nombres

  (define word (paradigmadocs "Word" (date 24 10 2021) encryptFn encryptFn))

  (define excel (paradigmadocs "Excel" (date 03 06 2021) encryptFn encryptFn))

  (define gDocs (paradigmadocs "GDoc" (date 10 12 2020) encryptFn encryptFn))

  ; se registran distintas cantidad de usuarios para los 3 paradigmadocs

  (define word1 (register (register word (date 05 07 2021) "nico" "1234") (date 05 07 2021) "raul" "4321"))

  (define excel1 (register (register excel (date 05 07 2021) "nico" "1234") (date 05 07 2021) "Loki" "4321"))

  (define gDocs1 (register (register (register gDocs (date 05 07 2021) "nico" "1234") (date 05 07 2021) "Loki" "4321") (date 24 09 2021) "cody" "5678"))

  ; se crean documentos en los paradigmadocs mediante el login y el create, en gDocs se hace un intento de prueba con usuario erroneo

  (define word2 ((login word1 "nico" "1234" create) (date 02 03 2021) "Informe Nico" "introduccion..."))

  (define excel2 ((login excel1 "Loki" "4321" create) (date 02 03 2021) "Lista Loki" "1- MasterCat /n"))

  (define gDocs2 ((login gDocs1 "cody" "5678" create) (date 02 03 2021) "Revision Cody" "contenido Revision"))

  (define gDocsError ((login gDocs1 "cody" "1234" create) (date 02 03 2021) "Revision Cody" "contenido Revision"))

  ; los dueños de documentos en los paradigmadocs les dan accesos a todos los demas usuarios registrados

  (define word3 ((login word2 "nico" "1234" share) 1 (access "raul" #\r)))

```
(define excel3 ((login excel2 "Loki" "4321" share) 1 (access "nico" #\w)))

(define gDocs3 ((login gDocs2 "cody" "5678" share) 1 (access "Loki" #\r)
(access "nico" #\c)))

(define gDocsError1 ((login gDocs2 "cody" "5678" share) 1 (access
"Loki" #\r) (access "nico" #\c)))

; los dueños de documentos o usuarios con permiso de escritura
añaden contenido al final de los documentos registrados

(define word4 ((login word3 "nico" "1234" add) 1 (date 12 23 2021) " /n
Desarrollo..."))

(define excel4 ((login excel3 "nico" "1234" add) 1 (date 12 23 2021)
"2-Atun"))

(define gDocs4 ((login gDocs3 "cody" "5678" add) 1 (date 12 23 2021) "
/nConclusion Revision: "))

(define gDocsError2 ((login gDocs3 "Loki" "4321" add) 1 (date 12 23
2021) " /nConclusion Revision: "))

; de los documentos creados y con texto añadido se vuelve a la version
anterior

(define word5 ((login word4 "nico" "1234" restoreVersion) 1 0))
(define excel5 ((login excel4 "Loki" "4321" restoreVersion) 1 0))
(define gDocs5 ((login gDocs4 "cody" "5678" restoreVersion) 1 0))

; se revocan todos los accesos a los documentos
(define word6 (login word5 "nico" "1234" revokeAllAccesses))
(define excel6 (login excel5 "Loki" "4321" revokeAllAccesses))
(define gDocs6 (login gDocs5 "cody" "5678" revokeAllAccesses))


;se crean mas documentos en las distintas plataformas con el fin de
probar la funcion search (se restaunran los accesos anteriores)

(define word7 ((login word5 "nico" "1234" create) (date 12 12 2021)
"segundo docucumento" "Prueba de segundo documento")  )

(define excel7 ((login excel5 "nico" "1234" create) (date 07 10 2021)
"prueba Nico" "contenido nico")  )

(define gDocs7 ((login gDocs5 "Loki" "4321" create) (date 08 08 2021)
"Documento Loki" "contenido de loki")  )
```

```
; distintas usuarios buscan documentos mediante una palabra clave
(define search1 ((login word7 "nico" "1234" search) "tro"))
(define search2 ((login excel7 "nico" "1234" search) "Master"))
(define search3 ((login gDocs7 "Loki" "4321" search) "contenido"))
(define searchNull ((login gDocs7 "Loki" "4321" search) "conclusion"))

;se imprime la informacion de los paradigmadocs creados y luego la
relacionada a ciertos usuarios

(display (paradigmadocs->string word7))
(display (paradigmadocs->string excel7))
(display (paradigmadocs->string gDocs7))
(display (login word7 "nico" "1234" paradigmadocs->string))
(display (login excel7 "Loki" "4321" paradigmadocs->string))
(display (login gDocs7 "cody" "5678" paradigmadocs->string))
(display (login gDocs7 "nico" "1234" paradigmadocs->string))
```