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

INFORME DE LABORATORIO 1: SIMULACIÓN DE UNA PLATAFORMA ESTILO "GOOGLE DOCS" EN SCHEME

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

04 de noviembre 2021

Tabla de Contenidos

1. INTRODUCCIÓN

El presente informe corresponde al laboratorio número 1 del curso de Paradigmas de Programación. Este primer laboratorio corresponde al Paradigma Funcional, donde se utiliza el lenguaje de programación Scheme a través del compilador Dr. Racket. 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 una plataforma de edición de texto colaborativo, como, por ejemplo, Google Docs. Esta plataforma 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 la plataforma, entre otras cosas. Para implementar esto, se deben tener algunos elementos en cuenta:

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

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.

Operaciones: Son las funciones 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 Funcional, a través de la programación en Scheme y el compilador Dr. Racket.

1.2 DESCRIPCIÓN DEL PARADIGMA

El Paradigma Funcional, tal como su nombre lo dice, tiene como unidad de abstracción y programación básica a las funciones.

Una función corresponde a una transformación de elementos, donde se tiene al dominio, que son los elementos de entrada y que serán transformados por el proceso de la función y el recorrido, que es el elemento de salida de la función una vez se ha realizado la transformación.

Tal como se menciona antes, el paradigma solo trabaja con funciones, por lo que no existe el concepto de variable actualizable en este paradigma. En el caso de este trabajo, se hace uso del lenguaje de programación funcional Scheme, aunque cabe destacar de que este no es un lenguaje puro de programación funcional. Un lenguaje puro de programación funcional corresponde a aquel lenguaje que solo tiene aspectos de la programación funcional, el cual no es el caso de Scheme ya que este tiene aspectos de la programación imperativa.

El paradigma, a través de la programación declarativa y al contrario de otros paradigmas, se basa en el ¿QUÉ? y no en el ¿CÓMO?, lo que se traduce en que el Paradigma Funcional se enfoca en el resultado y no en cómo se llega a este, lo cual conlleva un mayor nivel de abstracción. El paradigma trabaja bajo el Cálculo Lambda, el cual es un tipo de notación formal introducido para el estudio de funciones.

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

Funciones Anónimas: Provienen directamente del Cálculo Lambda, donde estas funciones se expresan sin nombre, bajo una entrada y una transformación de esta que se aplica a la función.

Composición de funciones: Consiste en una especie de operación donde dos funciones generan una tercera función, algo bastante útil cuando se tienen funciones que se complementan entre sí.

Recursividad: Es fundamental en este paradigma, pues permite realizar una gran cantidad de operaciones y procesos donde este se van utilizando soluciones pequeñas del problema. Existen tres tipos de recursión en este Paradigma: recursión natural, recursión de Cola y recursión Arbolea, pero para este proyecto se utilizan los primeros dos tipos.

Funciones de orden superior: Corresponden a aquellas funciones que reciben como dominio a una función o bien dan como resultado a otra función. El mejor ejemplo corresponde a la composición de funciones.

Currificación: Es la evaluación de una función de “n” argumentos, transformada a la evaluación de una secuencia de funciones de un argumento. En la currificación se ocupan funciones anónimas bajo el concepto de funciones de orden superior.

1.3 OBJETIVOS

Como objetivos del proyecto se tiene aprender sobre el Paradigma y la programación funcional, para así obtener la habilidad de programar de otra forma distinta a la que se tiene costumbre actualmente. Otro objetivo es programar correctamente en Scheme y aprender a utilizar las herramientas de Dr. Racket para completar el proyecto de laboratorio y así

poder tener una base para los futuros laboratorios y otros proyectos que en un futuro se desarrollen con la Programación Funcional.

2. DESARROLLO

2.1 ANALISIS 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 la plataforma a través de la consola de Dr. Racket. Entendiendo esto, se tienen algunos elementos fundamentales en el problema:

Primero, la plataforma en sí, la 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. La plataforma registra todos los cambios que se realizan exitosamente a través de una operación. Teniendo en cuenta esto, se requiere representar a la plataforma en Scheme, donde se puede representar de la siguiente manera:

Plataforma: Lista que contiene a un string, una fecha, la función encryptFunction, la función decryptFunction y tres listas donde se irán guardando los usuarios registrados, el usuario activo y los documentos respectivamente. (string X fecha X encryptFunction X decryptFunction X list X list X list)

Las funciones **encryptFunction** y **decryptFunction** quedan guardadas en la plataforma y ambas funciones encriptan y desencriptan un texto respectivamente. Los passwords de los usuarios junto con los contenidos de los documentos deben ser encriptados.

Luego, se tienen otros elementos importantes que se guardan en la plataforma:

- **Usuarios:** Corresponden a aquellos que interactúan con la plataforma, registrándose en esta 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 la plataforma 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 Scheme, se pueden representar de la siguiente manera:

- **Usuarios:** Una lista que contiene una fecha y dos strings (fecha X string X string). Cabe destacar que, para el caso del usuario activo, solo se guarda en una lista un string.
- **Documentos:** Una lista que contiene una lista que contiene tres strings, una fecha, dos listas y un integer (string X fecha X string X string X list X list X integer)
- **Permisos:** Una lista que contiene un string y un carácter (string X character)
- **Historial de versiones:** Una lista 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:

- **Register:** Verifica si un username no ha sido registrado. Si es así, registra al usuario en la plataforma.
- **Login:** Verifica las credenciales de un usuario registrado y si son correctas, deja al usuario como usuario activo (logueo correcto). Llama a las operaciones a continuación.
- **Create:** Crea un nuevo documento y lo guarda en la plataforma
- **Share:** Permite compartir un documento con otros usuarios registrados en la plataforma. Los permisos se guardan en el documento especifico.
- **Add:** Añade texto al final de la versión activa de un documento.
- **RestoreVersion:** Restaura una versión de un documento, guardada en el historial.
- **RevokeAllAccesses:** Elimina todos los permisos otorgados en un documento.
- **Search:** Permite buscar documentos (propios o que se tenga permisos) que contengan un texto en específico. Retorna una lista con los documentos encontrados.
- **Paradigmadocs->String:** Convierte el contenido de Paradigmadocs en un string.
- **Delete:** Permite eliminar texto en una versión activa de un documento.
- **SearchAndReplace:** Permite buscar y reemplaza un texto en una versión activa de un documento.
- **Applystyles:** Permite añadir estilos a un texto en específico de un documento.
- **Comment:** Permite añadir comentarios a un texto en específico de un documento.
- **EncryptFn y DecryptFn:** Permiten encriptan y desencriptar texto respectivamente.

Cabe destacar de que todas las funciones desde Create hasta Comment requieren de un login correcto para obtener el resultado esperado, a excepción de Paradigmadocs->String, donde esta operación de igual forma retorna todo el contenido de Paradigmadocs si no hubo un Login exitoso. Paradigmadocs->string retorna un string que debe ser utilizado a través de la función “display” para que sea visualizado correctamente.

Se pide que cada TDA (Tipo de Dato Abstracto) tenga un archivo único, donde solo se implemente lo necesario para cada TDA y que el archivo principal contenga las funciones obligatorias y opcionales, donde en este se llamen a las funciones de los TDAs creados.

2.2 DISEÑO DE LA SOLUCIÓN

Para diseñar la solución, no solo es necesario utilizar los tipos de datos nativos de Scheme, sino que también se deben crear tipos de datos específicos, a través de lo que se conoce como TDA, utilizando la siguiente estructura: Representación, Constructores, Funciones de Pertenencia, Selectores, Modificadores y otras funciones asociadas al TDA.

Los TDAs creados son utilizados para la construcción de las operaciones de otros TDAs y para las operaciones obligatorias y opcionales. Los TDAs más importantes son:

2.2.1.1 TDA PARADIGMADOCS

- Representación: Una lista que contiene: un string, una fecha, las funciones EncryptFunction y DecryptFunction, una lista para guardar a los usuarios registrados, una lista para guardar al usuario activo y una lista para guardar a los documentos.
- Constructor: Dados un nombre, una fecha y las funciones EncryptFunction y DecryptFunction, se crea una lista que tiene los elementos anteriormente mencionados junto con tres listas vacías.
- Función de Pertenencia, Selectores, Modificadores y Otras Funciones: *Ver Tabla N°1 en Anexos, P. 13.*

2.2.1.2 TDA USER

- Representación: Una lista que contiene una fecha y dos strings.
- Constructor: Dados una fecha de registro, un username y un password, se crea una lista que contiene estos tres elementos.
- Función de Pertenencia, Selectores y Otras Funciones: *Ver Tabla 2 en Anexos, P. 14.*

No se usan modificadores para este TDA.

2.2.1.3 TDA DOCUMENTO

- Representación: Una lista que contiene tres strings, una fecha, dos listas y un integer
- Constructor: Dados un autor, una fecha, un nombre, un texto y un ID, se crea una lista que contiene las entradas anteriormente mencionadas junto con las dos listas. La segunda lista corresponde a un Historial de Versiones
- Funciones de Pertenencia, Selectores, Modificadores y Otras funciones: *Ver Tabla N°3 en Anexos, P. 14.*

2.2.1.4 OTROS TDAS

Los TDAs más pequeños, pero que se utilizan en otros TDAs son:

- **TDA Fecha:** Representado como una lista que contiene tres integers
- **TDA Access:** Representado como una lista que contiene un conjunto de elentos
- **TDA Historial:** Representado como una lista que contiene una fecha, un string y un integer

- **TDA Permiso:** Representado como una lista que contiene un string y un carácter.

Ver la Figura N°1 en ANEXOS para poder tener más información sobre el uso de TDAs en otros TDAs. P.16.

2.2.2 OPERACIONES OBLIGATORIAS

REGISTER: Se implementa una función la cual permite verificar si un User está registrado en la plataforma y si no es así, se registra al User. El password se encripta usando encryptFunction, la cual esta guardada en Paradigmadocs. La verificación se hace a través del uso de una función con Recursión Natural (registerNatural).

LOGIN: Se implementa una función Recursiva de Cola, la cual permite loguear a un User si corresponde a uno de los Users registrados en la plataforma. Si el login es exitoso, se llama a la operación con un Paradigmadocs que contiene a un User activo. Caso contrario, se llama a la operación con Paradigmadocs sin modificaciones. Las siguientes funciones requieren del uso de Login para que su proceso sea exitoso, donde se le da un Paradigmadocs y recibe los demás parámetros gracias a la semi currificación.

CREATE: Se implementa una función semi currificada, la que crea un Documento cuyo autor corresponde al User activo. A través de una función se verifica el largo de la lista Documentos para así asignarle el ID correcto al documento. Se agrega a Paradigmadocs.

SHARE: Se implementa una función Recursiva de Cola y semi currificada, donde se recibe una lista utilizando el TDA Access y se verifica que el ID corresponda a uno de los documentos de Paradigmadocs. Se filtran los permisos y luego se agregan al Documento.

ADD: Se implementa una función Recursiva de Cola y semi currificada, donde se verifica el ID corresponda a uno de los documentos de Paradigmadocs y a través de la recursividad se verifica si el User activo puede escribir (es autor o tiene permisos). Se agrega el texto al final de la versión activa, creando una versión nueva y esta es encriptada.

RESTOREVERSION: Se implementa una función Recursiva de Cola y semi currificada, donde se verifica que el ID corresponda a uno de los documentos de Paradigmadocs. Luego se verifica que el ID del historial corresponda a una de las versiones y se reemplaza la versión activa del Documento.

REVOKEALLACCESES: Se implementa una función Recursiva de Cola, donde se utiliza la recursión para encontrar todos los documentos donde el User activo es el autor y luego, se utiliza la función MAP para eliminar todos los permisos de los documentos encontrados.

SEARCH: Se implementa una función Recursiva de Cola y semi currificada, donde primero, utilizando recursión se encuentran todos los Documentos donde el User es autor y tiene permisos. Luego se utiliza la función FILTER para encontrar aquellos documentos que tienen el texto correspondiente. Se retorna una lista de documentos.

PARADIGMADOCS->STRING: Se implementa una función donde se utiliza la función MAP junto con funciones creadas en los TDAs y la función STRING-JOIN para transformar los elementos de Paradigmadocs en strings. Se tiene como resultado un string que debe ser utilizado con la función DISPLAY para ser visualizado correctamente.

2.2.3 OPERACIONES OPCIONALES

DELETE: Se implementa una función Recursiva de Cola y semi currificada, bastante similar a Add pero con la diferencia de que se elimina texto de un documento. Se utiliza la función STRING->LIST para poder eliminar correctamente la cantidad de caracteres deseado.

SEARCHANDREPLACE: Se implementa una función Recursiva de Cola y semi currificada, donde primero se verifica si el ID corresponde a uno de los documentos de Paradigmadocs, luego se utiliza la función STRING-REPLACE para poder reemplazar correctamente el texto encontrado.

APPLYSTYLES: Se implementa una función Recursiva de Cola y semi currificada, donde primero se verifica si el ID corresponde a uno de los documentos de Paradigmadocs, luego se filtra la lista de estilos y se agrega al texto deseado.

COMMENT: Se implementa una función Recursiva de cola y semi currificada, donde primero se verifica si el ID corresponde a uno de los documentos de Paradigmadocs, luego se utiliza la función STRING-REPLACE para poder encontrar el texto correcto y comentarlo. Los comentarios se simbolizan con &C&(“Comentario”)&C&.

ENCRYPTFN Y DECRYPTFN: Se implementa una función que hace de encriptación y desencriptación de texto. Intercambia un texto por otro, diferenciando por tamaño de letra y por letras, números y símbolos. (No requiere de Login para ser utilizada)

2.3 ASPECTOS DE IMPLEMENTACIÓN

2.3.1 COMPILADOR

Para este proyecto es necesario el compilador Dr. Racket, específicamente de versión 6.11 o superior. Dr Racket tiene bastantes herramientas utiles, como por ejemplo debug, que permite revisar pasó a pasó el cambio del Stack del código y a través de esto encontrar errores. Se pueden utilizar todo tipo de funciones de Scheme y Racket.

La implementación se baja en el trabajo con listas, sin embargo, se espera que no se utilicen funciones como CAR o CDR fuera de la elaboración de TDAs, ya que estas son funciones del TDA Lista, por lo que se deben encapsular dentro de los TDAs creados.

2.3.2 ESTRUCTURA DEL CODIGO

Para que funcione el estructurar el código en archivos independientes y poder utilizar funciones de ciertos TDAs en otros archivos .rkt, se hace uso de las funciones REQUIRE y PROVIDE, las cuales permiten importar y exportar respectivamente.

Para todos los TDAs, se hace uso de "(provide (all-defined-out))", lo cual permite que se puedan utilizar todas las funciones creadas dentro de ese archivo en otro archivo, sin necesidad de estar importando funciones una tras una. Para el archivo main, se hace uso de todos los TDAs creados importándolos a través de REQUIRE.

En total, se tienen 8 archivos: "TDAParadigmadocs.rkt", "TDAUser.rkt", "TDADocumento.rkt", "TDAHistorial.rkt", "TDAAccess.rkt", "TDAPermiso.rkt", "TDAFecha.rkt" y "main.rkt" (cada uno con la identificación solicitada).

Ver Figura N°1 en Anexos para ver en que archivos de importan otros archivos.

2.4 INSTRUCCIONES DE USO

2.4.1 EJEMPLOS DE USO

Primero, se debe verificar que se tengan los ocho archivos en una misma carpeta, ya que, de lo contrario, el archivo main no se podrá ejecutar si falta un TDA. Luego de eso, se puede compilar, ejecutando el programa a través de la opción "Run".

Una vez ejecutado, en el archivo main se tienen varios ejemplos para cada una de las operaciones. Para que estos ejemplos puedan funcionar correctamente, se requiere tener creado un Paradigmadocs, el cual por defecto ya hay uno creado dentro del archivo de TDA Paradigmadocs. Cada ejemplo tiene un "nombre" y para ejecutarlo solo se debe escribir el nombre del ejemplo en la consola y apretar enter para ejecutar la operación.

En el caso de querer probar el código sin utilizar los ejemplos anteriormente mencionados, se deben seguir los siguientes pasos:

1. Crear un Paradigmadocs. Esto se hace a través de la función paradigmadocs.
2. Luego de eso, se pueden ejecutar las funciones de main sin inconvenientes, sin embargo, la gran mayoría de estas requiere de usuarios registrados y de login. Primero, se puede registrar un usuario a través de la función register.
3. Posterior al registro de Users, ahora se puede utilizar la función login, donde se recibe una operación. Dependiendo de la operación, se debe aprovechar la semi currificacion para poder pasarle los parámetros a la operación escogida. Si el login es exitoso, a todas las operaciones se les pasa la plataforma con el usuario logueado. Repitiendo este paso se pueden utilizar todas las operaciones deseadas que estén dentro de main.

Ver en Anexos la Figura N°2, la cual es un ejemplo paso a paso con el procedimiento anterior para el caso de la función Create. P.16. Ver Figuras N°3 – N°5. P.17,

2.4.2 RESULTADOS ESPERADOS

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

Junto con lo anterior, se espera trabajar correctamente con listas y recursión, ya que son fundamentales para el desarrollo del proyecto. Por último, se espera que cada función haga su procedimiento correctamente y que los TDAs tengan representaciones correctas.

2.4.3 POSIBLES ERRORES

El único posible error que se puede producir en el programa es que al aplicar la función ApplyStyles por segunda vez con el mismo texto donde se aplicó anteriormente, el estilo se repetirá a pesar de que ya fue aplicado antes. Al hacer pruebas del programa, no se encuentra ningún otro error, funcionando correctamente con todas las funciones implementadas.

2.5 RESULTADOS Y AUTOEVALUACIÓN

2.5.1 RESULTADOS OBTENIDOS

Los resultados obtenidos fueron los esperados, ya que se logró crear las funciones obligatorias y la gran mayoría de las funciones opcionales. El programa es completamente funcional, incluyendo en casos de contraejemplos donde las entradas colocadas son erróneas. Solo se tiene un pequeño error el cual fue mencionado anteriormente, que lamentablemente no se pudo llegar a una solución concreta, pero aun así el programa no tira ningún error al ser ejecutado. Por lo tanto, se logró crear una simulación de una plataforma de edición 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 completaron 14/15 funciones, donde la única que no se realizó fue la última función opcional (CtrlY y CtrlZ).

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.

Para ver la tabla de Autoevaluación, *ver la Tabla N°4 en Anexos, P.18.*

Solo en la función ApplyStyles se elige 0.75, ya que se considera que a pesar de que funciona sin dar error, se tiene el error de los estilos repetidos en un texto, lo cual realmente no debería suceder. En todas las demás funciones y TDAs no se encontró errores, por lo que se considera de que funcionan el 100% de las veces.

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 Scheme y Racket 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 los TDAs, recursividad, funciones de orden superior, semi currificación, etc.

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 se tenía la currificación de las funciones aceptada por login dentro de la función login, pero eso causa que la función login deje de ser agnóstica, por lo que se decide aplicar la idea de colocar las semi currificaciones en los encabezados de las funciones y así evitar el problema mencionado anteriormente. Otra gran complicación era el tiempo, ya que este es un proyecto que requiere de bastante tiempo para su comprensión y para lograr realizar todo lo requerido, pero aún así, se pudo aprovechar bien e incluso se pudieron hacer funciones opcionales ya que sobró una gran cantidad de tiempo antes de la entrega.

Dejando lo anterior de lado, no hubo complicaciones de comprensión de conceptos o problemas con las herramientas como Git o Racket en general. Al contrario, gracias a este proyecto se pudieron aprender muchas herramientas y funciones de Racket y de Github que no se conocían antes, como, por ejemplo, las funciones MAP y FILTER, las cuales definitivamente serán útiles si en un futuro se desea volver a trabajar con la programación funcional. Se espera que este conocimiento también sea útil para el desarrollo de los próximos proyectos de laboratorio y en los próximos paradigmas que se verán en el curso.

4. BIBLIOGRAFÍA Y REFERENCIAS

1. Gonzales, R. (2021). “Proyecto Semestral de Laboratorio”. Paradigmas de Programación. Enunciado de Proyecto Online. Recuperado de: https://docs.google.com/document/d/1pORdJwp5WJ7V3DIAQik9B58llpdxpurQRVTKPZHOpS8/edit
2. Gonzales, R. (2021). “3 – P. Funcional”. Paradigmas de Programación. Material de clases Online. Recuperado de: https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=11
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. (Febrero 1998). “Report on the Algorithmic Language Scheme”. Informe / Estudio Online. Recuperado de: https://uvirtual.usach.cl/moodle/pluginfile.php/369069/mod_resource/content/2/r5rs.pdf
5. Flatt, M. y Bruce, R. (2021). “The Racket Guide’’. The Racket Reference. Documentación Online. Recuperado de: https://docs.racket-lang.org/guide/

5. ANEXOS

Tabla N°1: Funciones restantes del TDA Paradigmadocs.

Tipo de función	Nombre	Descripción
Función de Pertenencia	isParadigmadocs?	Verifica si un elemento corresponde a un Paradigmadocs
Selector	getNombrePdocs	Obtiene el nombre de un Paradigmadocs
Selector	getFechaPdocs	Obtiene la fecha de creación de un Paradigmadocs
Selector	getEncryptPdocs	Obtiene encryptFunction de Paradigmadocs
Selector	getDecryptPdocs	Obtiene decryptFunction de Paradigmadocs
Selector	getUsersPdocs	Obtiene la lista de Users registrados de Paradigmadocs
Selector	getUsersactivosPdocs	Obtiene la lista de Users activos de Paradigmadocs
Selector	obtenerActivoPdocs	Obtiene al User activo de la lista de Users activos de Paradigmadocs
Selector	getDocumentosPdocs	Obtiene la lista de Documentos de Paradigmadocs
Modificador	setUserPdocs	Agrega a un User a Paradigmadocs
Modificador	setUseractivosPdocs	Agrega a un User como activo a Paradigmadocs
Modificador	setDocumentoPdocs	Agrega un Documento a Paradigmadocs. Si hay un Documento con el mismo ID, lo reemplaza
Modificador	setRemoverActivoPdocs	Remueve a un User activo de Paradigmadocs
Modificador	setDocumentosAlternPdocs	Versión alternativa de setDocumentoPdocs, solo que no se remueve al User activo
Modificador	setDocumentosPermisosPdocs	Agrega Permisos a un Documento de Paradigmadocs
Modificador	setListaDocumentosPdocs	Elimina y vuelve a agregar la lista de Documentos, pero esta vez modificada, a Paradigmadocs
Otras Funciones	revisarUserActivoPdocs	Verifica si un user esta registrado y puede loguearse
Otras Funciones	definirID	Agrega un ID correcto a un Documento de Paradigmadocs
Otras Funciones	revisarUsernPdocs	Verifica si un User tiene el mismo username que otro User o un Autor de un Documento
Otras Funciones	filtrarPermisosPdocs	Filtra la lista de permisos para dejar los permisos solo de aquellos Users registrados en Paradigmadocs
Otras Funciones	obtenerDocumentosAutor	Obtiene todos los Documentos donde un User es Autor
Otras Funciones	obtenerDocumentosUser	Obtiene todos los Documentos donde un User NO es Autor
Otras Funciones	filtrarPorPermisos	Obtiene todos los Documentos donde un User no es autor, pero tiene permisos de escritura
Otras Funciones	encontrarDatosUsuarioPdocs	Obtiene el username y fecha de registro de un User, transformando todo a formato String
Otras Funciones	filtrarPorAccesos	Obtiene todos los Documentos donde un User tiene permisos
Otras Funciones	aplicarEstilos	Filtra y aplica estilos a un texto de un Documento

Tabla N°2: Funciones restantes del TDA User.

Tipo de función	Nombre	Descripción
Función de Pertenencia	isUser?	Verifica si un elemento corresponde a un User
Selector	getFechaUser	Obtiene la fecha de registro de un User
Selector	getUsernameUser	Obtiene el Username de un User
Selector	getPasswordUser	Obtiene el Password de un User
Selector	getPrimeroListUser	Obtiene el primer User de una lista de Users
Selector	getSiguientesListUser	Obtiene una lista de Users sin el primer User
Otras Funciones	registerNatural	Verifica si un User está en una lista de Users y utilizando recursión Natural devuelve una lista con el User agregado si es que no estaba.
Otras Funciones	revisarUsuarioPdocs	Verifica si un User se encuentra en una lista de Users
Otras Funciones	usersIguales?	Verifica si dos Users tienen el mismo username
Otras Funciones	verificarUsersUser	Verifica si dos Users son exactamente iguales
Otras Funciones	usernamesIguales?	Versión alternativa de usersIguales? Pero verifica si el nombre de un User corresponde a un string a comparar
Otras Funciones	userToString	Transforma toda la información de un User (incluido el password) a string.

Tabla N°3: Funciones restantes del TDA Documento.

Tipo de función	Nombre	Descripción
Función de Pertenencia	isDocumento?	Verifica si un elemento corresponde a un Documento
Selector	getAutorDocumento	Obtiene el autor de un Documento
Selector	getFechaDocumento	Obtiene la fecha de creación de un Documento
Selector	getNombreDocumento	Obtiene el nombre de un Documento
Selector	getContenidoDocumento	Obtiene el contenido de un Documento
Selector	getPermisosDocumento	Obtiene la lista de permisos de un Documento
Selector	getHistorialDocumento	Obtiene la lista del Historial de versiones de un Documento
Selector	getIDDocumento	Obtiene el ID de un Documento
Selector	getPrimeroListDocumento	Obtiene el primer Documento de una lista de Documentos
Selector	getSiguientesListDocumento	Obtiene una lista de Documentos sin el primer Documento
Modificador	setContenidoDocumento	Agrega un nuevo contenido a un Documento, dependiendo si este es distinto al que ya está como versión activa. También depende de un booleano de si se agrega al final de la versión existente o se agrega como texto totalmente nuevo.
Modificador	setPermisosDocumento	Agrega permisos a la lista de permisos de un Documento
Otras Funciones	unirListasDocumentos	Une dos listas de Documentos
Otras Funciones	verificarIDs	Verifica si un Documento tiene un ID en especifico

Otras Funciones	encontrarIDs	Encuentra un Documento que tiene un ID en especifico en una lista de Documentos
Otras Funciones	obtenerIDHistorial	Obtiene el ID correcto para una versión del Historial de un Documento
Otras Funciones	restaurarVer	Restaura la versión de un Documento (si esta existe)
Otras Funciones	eliminarPermisos	Elimina todos los permisos de un Documento
Otras Funciones	addTextoSearch	Agrega un texto en especifico a todos los Documentos de una lista de Documentos, esto para ser utilizado en la función Search
Otras Funciones	encontrarTexto	Busca un texto en especifico en la versión activa de un Documento
Otras Funciones	eliminarRastro	Elimina el rastro de addTextoSearch
Otras Funciones	documentoToString	Transforma todo el contenido de un Documento (incluyendo Permisos e Historial) a String
Otras Funciones	deleteCharsDoc	Elimina una cantidad determinada de caracteres a la versión activa de un Documento
Otras Funciones	mezclarLetras	Cambia un texto por otro
Otras Funciones	obtenerSinComentario	Obtiene la última versión de un Documento que no tiene comentarios.

Figura N°1: Diagrama de importación de TDAs.

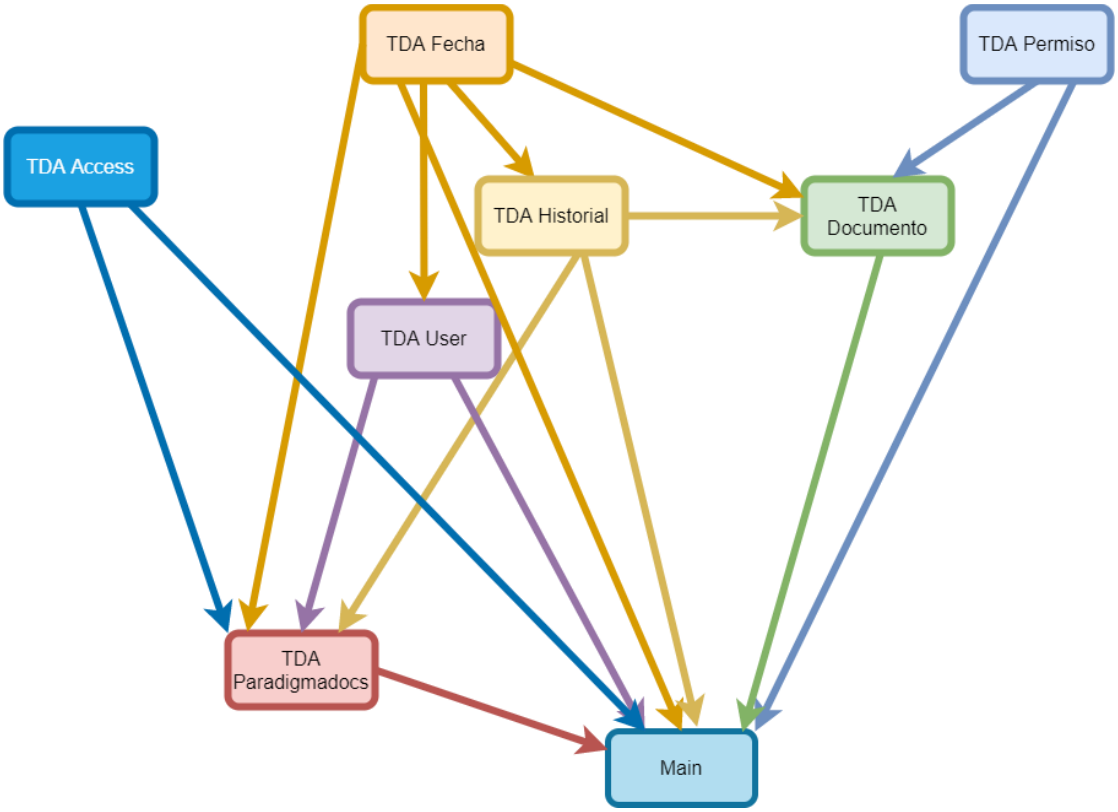

Figura N°2: Ejemplo número 1: Paso a paso para utilizar la función Create.

```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define nuevoGDocs (paradigmadocs "Test" (date 04 11 2021) encryptFunction decryptFunction))
> nuevoGDocs ; Se crea un nuevo Paradigmadocs
'("Test" (4 11 2021) #<procedure:encryptFunction> #<procedure:decryptFunction> () () ())
> (define nuevogDocs2 (register nuevoGDocs (date 04 11 2021) "user1" "pass1"))

> nuevogDocs2 ; Se registra un user al Paradigmadocs guardado anteriormente
'("Test"
  (4 11 2021)
  #<procedure:encryptFunction>
  #<procedure:decryptFunction>
  (((4 11 2021) "user1" "1ssap"))
  ()
  ())
> (define nuevogDocs3 ((login nuevogDocs2 "user1" "pass1" create) (date 04 11 2021) "doc0" "doc0"))

> nuevogDocs3 ; Se crea un nuevo documento, utilizando la funcion Login y la funcion Create
'("Test"
  (4 11 2021)
  #<procedure:encryptFunction>
  #<procedure:decryptFunction>
  (((4 11 2021) "user1" "1ssap"))
  ()
  (("user1" (4 11 2021) "doc0" "0cod" () (((4 11 2021) "0cod" 0)) 0)))
```

Nota: Al final del archivo main están demás ejemplos listos para ejecutar. Las siguientes figuras muestran algunos de estos.

Figura N°3: Ejemplo "listo para ejecutar" de la función Create, en un Paradigmadocs que ya tiene 5 users registrados y un documento creado.

```
312 ; Se loguea al user2 correctamente y se crea un segundo documento
313 (define gDocs2e ((login gDocs2d "user2" "pass2" create) (date 26 10 2021) "doc1" "doc1"))
314
```

```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> gDocs2e
'("gDocs"
  (25 10 2021)
  #<procedure:encryptFunction>
  #<procedure:decryptFunction>
  (((25 10 2021) "user1" "1ssap")
   ((25 10 2021) "user2" "2ssap")
   ((25 10 2021) "user3" "3ssap")
   ((25 10 2021) "user4" "4ssap")
   ((26 10 2021) "user5" "5ssap"))
  ()
  (("user2" (26 10 2021) "doc1" "1cod" () (((26 10 2021) "1cod" 0)) 1)
   ("user1" (25 10 2021) "doc0" "0cod" () (((25 10 2021) "0cod" 0)) 0)))
```

Figura N°4: Ejemplo "listo para ejecutar" de la función Share.

```
333 ; User 2 se loguea correctamente y le da acceso de lectura a user 1 y a acceso de escritura a user 3 para
334 (define gDocs3e ((login gDocs3d "user2" "pass2" share) 1 (access "user1" #\r) (access "user3" #\w)))
335
```

```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> gDocs3e
'("gDocs"
  (25 10 2021)
  #<procedure:encryptFunction>
  #<procedure:decryptFunction>
  (((25 10 2021) "user1" "1ssap")
   ((25 10 2021) "user2" "2ssap")
   ((25 10 2021) "user3" "3ssap")
   ((25 10 2021) "user4" "4ssap")
   ((26 10 2021) "user5" "5ssap"))
  ()
  (("user2" (26 10 2021) "doc1" "1cod" (("user1" #\r) ("user3" #\w)) (((26 10 2021) "1cod" 0)) 1)
   ("user1" (25 10 2021) "doc0" "0cod" (("user2" #\r)) (((25 10 2021) "0cod" 0)) 0)
   ("user1" (26 10 2021) "doc2" "2cod" (("user3" #\w)) (((26 10 2021) "2cod" 0)) 2)))
```

Figura N°5: Ejemplo "listo para ejecutar" de la función Paradigmadocs->String.

```
413
414 ; Nadie se loguea y se utiliza paradigmadocs->string
415 (define gDocs8f (paradigmadocs->string gDocs5e))
```

```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> gDocs8f
" Nombre de la plataforma: gDocs \n Fecha de creacion: 25 de Octubre de 2021 \n Usuarios registrados en la
plataforma:\n ---------\n Username: user1 \n Password: 1ssap \n Fecha de registro: 25 de Octubre de 2021 \n
---------\n Username: user2 \n Password: 2ssap \n Fecha de registro: 25 de Octubre de 2021 \n ---------\n
Username: user3 \n Password: 3ssap \n Fecha de registro: 25 de Octubre de 2021 \n ---------\n Username: user4 \n
Password: 4ssap \n Fecha de registro: 25 de Octubre de 2021 \n ---------\n Username: user5 \n Password: 5ssap \n
Fecha de registro: 26 de Octubre de 2021 \n \n Documentos creados en la plataforma:\n ---------\n Autor: user2
\n Fecha de creacion: 26 de Octubre de 2021 \n Nombre de Documento: doc1 \n Contenido Documento (Version
Actual): doc1 \n Permisos: \n user1 tiene permiso de lectura \n  user3 tiene permiso de escritura \n  Historial
de versiones: \n Version N° 1 : * doc1 h * version guardada el dia 27 de Octubre de 2021 \n Version N° 0 : *
doc1 * version guardada el dia 26 de Octubre de 2021 \n ID Documento: 1 \n ---------\n Autor: user1 \n Fecha de
creacion: 25 de Octubre de 2021 \n Nombre de Documento: doc0 \n Contenido Documento (Version Actual): doc0 \n
Permisos: \n user2 tiene permiso de lectura \n  Historial de versiones: \n Version N° 1 : * doc0 t * version
guardada el dia 27 de Octubre de 2021 \n Version N° 0 : * doc0 * version guardada el dia 25 de Octubre de 2021
\n ID Documento: 0 \n ---------\n Autor: user1 \n Fecha de creacion: 26 de Octubre de 2021 \n Nombre de
Documento: doc2 \n Contenido Documento (Version Actual): doc2 \n Permisos: \n user3 tiene permiso de escritura
\n  Historial de versiones: \n Version N° 0 : * doc2 * version guardada el dia 26 de Octubre de 2021 \n ID
Documento: 2 \n"
```

Tabla N°4: Autoevaluación de los requerimientos funcionales

Requerimientos Funcionales	Evaluación
TDAs	1
Register	1
Login	1
Create	1
Share	1
Add	1
RestoreVersion	1
RevokeAllAccesses	1
Search	1
Paradigmadocs->String	1
Delete	1
SearchAndReplace	1
ApplyStyles	0.75
Comment	1
EncryptFn y DecryptFn	1