UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

SEDE VIÑA DEL MAR – JOSÉ MIGUEL CARRERA

BANCO DE CONTRASEÑAS
PARA SISTEMA OPERATIVO ANDROID VERSIÓN 4.1

Trabajo de Titulación para optar al Título de
Técnico Universitario en INFORMÁTICA

Alumno:
Benjamín Cristóbal Díaz Eyssautier

Profesora Guía:
Catherine Gómez Barrera

2018

DEDICATORIA

Las decisiones que tomamos en nuestra vida definen nuestra historia, y en este momento, me gustaría dedicar parte de mi historia, la obtención de mi primer título universitario, a todas las personas que incondicionalmente estuvieron a mi lado, mis padres, que, a pesar de las decisiones que he tomado, siempre me han apoyado en todo lo posible, emocionalmente, espiritualmente y económicamente, mi hermana que, a pesar de nuestras diferencias, siempre ha sido una fuente de alegría y risas, y a mi querida pareja, quien me ha empujado a crecer como persona y me ha motivado, día a día, a tomar nuevos desafíos.

RESUMEN

KEYWORDS: ADMINISTRADOR DE CONTRASEÑAS – ANDROID STUDIO – SISTEMA OPERATIVO ANDROID 4.1 JELLY BEAN – INTEGRACIÓN CON API DE GOOGLE DRIVE – MOTOR DE BASE DE DATOS SQLITE – METODOLOGÍAS CRIPTOGRÁFICAS

En la sociedad actual, no es extraño escuchar noticias donde gran cantidad de personas son víctimas de accesos no autorizados a sus cuentas de diversos servicios, o donde sistemas informáticos, muy populares, contienen vulnerabilidades que se traducen en el robo de datos sensibles de los usuarios que los utilizan, debido a esto, es tarea de todos el prevenir o reducir los posibles daños que se generan frente a situaciones de este estilo.

Entre las recomendaciones que se suelen mencionar, para reducir los riesgos de usar sistemas informáticos, uno puede encontrar que se debe utilizar una contraseña distinta para cada servicio, las contraseñas se deben cambiar de manera frecuente, y la composición de dichas contraseñas no deben ser predecibles, por lo cual, se desarrolla el siguiente sistema informático, con el objetivo de facilitar la puesta en práctica de las recomendaciones mencionadas.

El sistema almacenará las cuentas que el usuario posea, junto a sus respectivos historiales de contraseñas, las clasificará por categorías, de acuerdo con lo requerido por el usuario, y entregará herramientas para informar la necesidad de actualizar contraseñas, para generar contraseñas no predecibles, para respaldar en la nube la base de datos, y para mantener toda la información sensible segura, mediante metodologías criptográficas.

Conocido por el nombre de “Banco de Contraseñas”, el sistema informático se desarrolla utilizando la IDE de Android Studio, con lenguaje de programación Java y diseño de interfaces mediante XML, además utilizará SQLite como gestor de base de datos, y será ejecutado en dispositivos móviles, con sistema operativo Android, versión 4.1 o superior.

El contenido, del presente documento, se divide en los tres capítulos siguientes:

- Capítulo 1: Descripción de la situación actual, problemas detectados, y descripción del sistema propuesto.
- Capítulo 2: Medio ambiente computacional, herramientas de software a utilizar, y descripción de los archivos.
- Capítulo 3: Estructura del sistema, diagrama modular y de menús, y descripción de los programas.

En conclusión, para el desarrollo de un sistema informático íntegro, es necesario llevar a cabo cada una de las etapas que pertenecen al ciclo de vida de éste, desde el análisis y diseño, hasta la implementación y mantenimiento. En el caso de este trabajo de título, se analizan los problemas que afectan a personas que tienen la necesidad de identificarse en servicios informáticos, se diseña una solución que satisfaga los requerimientos encontrados, se construye dicha solución, y se implementa para satisfacer la problemática inicial.

ÍNDICE

ÍNDICE DE GRÁFICOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

No solo las empresas e instituciones requieren de sistemas informáticos para solucionar la gran variedad de problemas que surgen en base a los procesos que llevan a cabo, también existe un gran mercado para sistemas informáticos que solucionan problemas de personas, que no necesariamente pertenecen a una comunidad en específico.

Dicho lo anterior, y a pesar de no existir una empresa en específico a quien se le está resolviendo una falencia, el siguiente trabajo de título consiste en el desarrollo de un sistema informático que da solución a problemáticas que cualquier persona, que utilice sistemas informáticos que requieren de algún tipo de autentificación, puede verse afectada.

Comenzando por la etapa de análisis, donde, en vez de observar la situación actual de una empresa, se observa la situación actual en la que se encuentra un mercado que la aplicación puede satisfacer, se encuentra que las personas no siguen las prácticas de seguridad informática que los expertos recomiendan en la actualidad, y que las soluciones existentes, a dichos problemas, no integran todas las características que un usuario puede desear.

Una vez reconocidos dichos problemas, se procede a idear un sistema que ayude, de una manera u otra, a reducir la cantidad de personas que se ven afectada por dichas problemáticas, lo que da como resultado el objetivo principal del proyecto.

Luego se continua con el diseño del sistema, definiendo aspectos más técnicos, como el dónde se implementará el sistema, con qué tecnologías se desarrollará, y qué modelo de datos se deberá construir para satisfacer las necesidades.

Para terminar con la etapa de construcción y pruebas, la cual da como resultado un sistema informático que cumple todas las características definidas durante el diseño y satisface todas las necesidades encontradas durante el análisis.

Gracias a esto, en las próximas páginas se documenta todas las distintas actividades que se llevaron a cabo para permitir el desarrollo del sistema informático en cuestión, exceptuando por las etapas de implementación y mantenimiento, las cuales no se descartan para un futuro desarrollo, con el objetivo de demostrar que el alumno posee los requerimientos que la carrera Técnico Universitario en Informática le solicita, además de poder ser usado como guía para cualquier otro alumno que desea implementar un sistema fuera de lo considerado normal, enfocado en tecnologías móviles y metodologías criptográficas.

CAPÍTULO 1: ASPECTOS RELEVANTES DEL DISEÑO LÓGICO

1. ASPECTOS RELEVANTES DEL DISEÑO LÓGICO

1.1. DESCRIPCIÓN DE LA SITUACIÓN ACTUAL

Es normal, para un usuario de servicios informáticos, la utilización de nombres de usuarios y contraseñas, los cuales van variando de un servicio a otro. Debido a la gran cantidad de servicios informáticos en existencia, el usuario se acostumbra a utilizar la misma contraseña, o unas pocas contraseñas, para todas las diversas cuentas que posee, además dicha contraseña, en muchas oportunidades, está conformada por cadenas de caracteres predecibles, al igual como es mantenida durante largos periodos de tiempo. En otros casos, se puede observar que los usuarios utilizan herramientas informáticas para ayudarse con la tarea de administrar sus datos de identificación, herramientas tales como 1Password, LastPass, Enpass o Dashlane, las cuales fueron identificadas como las mejores herramientas para administrar contraseñas, de acuerdo con el artículo “Best Password Manager For Android”, escrito por Andrew Martonik, en abril del 2017, para la plataforma Android Central.

A continuación, veremos las principales características de cada una de ellas:

1Password:

- Es una herramienta de pago que tiene una *versión individual*, con un valor de 2.99 USD mensuales, y una *versión familiar* con un valor de 4.99 USD mensuales.
- Tiene la capacidad de sincronizarse con el servidor de 1Password, pero además se puede sincronizar con iCloud, Dropbox y WLAN Server.
- Un usuario puede sincronizar tantos dispositivos como quiera, incluyendo dispositivos Macs, Windows PCs, iPhones, iPads y móviles Android.
- Cada usuario tiene acceso a una *bóveda personal*, donde puede almacenar todos los *ítems* que desea, y tiene la posibilidad de crear *bóvedas extras*, las cuales pueden ser compartidas con otros usuarios, de tal manera que los *ítems* creados en éstas puedan ser consultados por todos ellos.
- Los *ítems* que vaya creando el usuario, que pertenecerán a una de las *bóvedas* configuradas, tendrán que ser definidos como elementos de una *categoría*.
- Existen 18 *categorías* diferentes, entre las que se incluyen Login, Bank Account, Email Account, Credit Card, entre otros.

- Cada *ítem* se compone de una serie de *campos*, los cuales vienen definidos por defecto dependiendo de la *categoría* escogida, pero existe la posibilidad de agregar nuevos *campos* a un *ítem*, a la medida del usuario.
- Los *ítems* poseen un *campo* especial llamado *Tags*, el cual es utilizado para relacionar y agrupar *ítems* entre ellos.
- Los *campos* de cada *ítem* se pueden agrupar en *secciones*, definidas por el usuario.
- La aplicación posee un generador de contraseñas aleatorias, el cual tiene dos modalidades, componer la contraseña por caracteres o por palabras, si se elige la primera, se puede definir la cantidad total de caracteres, si incluirá dígitos, si incluirá símbolos y si se incluirán caracteres ambiguos, en la segunda modalidad, se puede definir la cantidad total de palabras, y el tipo de carácter a utilizar como separador de palabras, como por ejemplo el espacio, el punto o el guion bajo.
- Se tiene un menú rápido, con el cual se tiene acceso a todos los *ítems*, a los *ítems* favoritos, a los *ítems* asociados a cada una de las *categorías*, en un orden predeterminado, a los *ítems* asociados a cada uno de los *tags* creados por el usuario, a los *ítems* que la aplicación considera que necesitan una actualización, debido a fragilidad, duplicidad o vejez de contraseñas, y a los *ítems* que han sido mandados a la papelera.

<u>LastPass:</u>

- Es una herramienta gratuita, con la posibilidad de implementar nuevas funcionalidades al pagar por la *versión premium*, la cual tiene un valor de 12 USD anuales. Las funcionalidades *premium* incluyen quitar las publicidades de la aplicación, habilitar una versión de escritorio para el usuario, otorgar 1 GB de almacenamiento en la nube para archivos encriptados, entre otros.
- Tiene la capacidad de sincronizar múltiples dispositivos y generar respaldos en la nube, mediante la utilización de los servidores privados de LastPass.
- Cada elemento creado, llamado *sitio*, puede pertenecer a un *directorio*.
- Los *directorios* son creados, modificados y eliminados por el usuario, el único *directorio* que no se puede modificar es el *directorio (ninguno)*, el cual hace referencia a los *sitios* que no se encuentran asociados a un *directorio*.
- Los *sitios* creados se componen de los *campos* Nombre, Dirección web, Directorio, Nombre de usuario, Contraseña y Notas. Además, se podrá seleccionar si es un sitio favorito o no.
- Se tiene un generador de contraseñas, el cual crea aleatoriamente una cadena de acuerdo a la cantidad de caracteres que el usuario desee, además se puede

seleccionar si se compondrá por letras mayúsculas, minúsculas o números, en este último caso, se indica la cantidad mínima de números.

- Se tiene un menú rápido, con el cual se tiene acceso a todos los *sitios* creados por el usuario, a todas las *notas seguras*, a todos los *perfiles de formularios*, al generador de contraseñas, al administrador de carpetas compartidas, entre otros. Al acceder a todos los *sitios* creados, éstos quedan agrupados, y ordenados alfabéticamente, de acuerdo con el *directorio* asociado, comenzando por los favoritos y el *directorio (ninguno)*, y siguiéndoles, por orden alfabético, todos los demás *directorios* definidos por el usuario.

Enpass:

- Es una herramienta de pago con un valor único, por dispositivo móvil, de 9.99 USD, pero posee una versión gratuita que limita al usuario a solo almacenar un total de 20 *elementos*. La versión de escritorio, compatible con computadores macOS, Windows PCs y Linux, tiene un costo cero.
- Tiene la capacidad de sincronizarse con iCloud, Dropbox, OneDrive, Google Drive, Box o ownCloud/WebDAV.
- Cada *elemento* creado debe pertenecer a una de las *categorías* definidas por la aplicación.
- Existen 9 *categorías* en total, entre las que se incluyen Iniciar Sesión, Contraseña, Tarjeta de Crédito, entre otros.
- Independiente de la *categoría* que se elija, los *elementos* creados pueden ser organizados en *carpetas* definidas por el usuario, las cuales pueden ser marcadas para tener un *acceso directo* en el menú rápido.
- Los *elementos* creados se componen de una serie de *campos*, los cuales vienen definidos por defecto dependiendo de la *categoría* escogida, pero existe la posibilidad de agregar nuevos *campos*, eliminar *campos* existentes y ordenar los *campos* definidos, todo a la medida del usuario.
- Se tiene un generador de contraseñas, el cual crea aleatoriamente una cadena de acuerdo a la cantidad de caracteres, dígitos, símbolos y mayúsculas que el usuario desee.
- Se tiene un menú rápido, con el cual se tiene acceso a todos los *elementos*, a los *elementos* favoritos, al navegador de *carpetas*, a los *elementos* asociados a cada una de las nueve *categorías* existentes, en un orden predeterminado, y, para terminar, ordenados por fecha de creación, a cada uno de los *accesos directos*, definidos por el usuario, que direccionan a determinadas *carpetas*.

Dashlane:

- Es una herramienta gratuita, con la posibilidad de implementar nuevas funcionalidades al pagar por la *versión premium*, la cual tiene un valor de 39.99 USD anuales. Las funcionalidades *premium* incluyen, el crear respaldos en la nube, sincronizar múltiples dispositivos y acceso mediante navegador web.
- Los respaldos en la nube y la sincronización de dispositivos funcionan utilizando los servidores privados de Dashlane.
- Funciona en plataformas Windows PC, Mac, iOS y Android.
- Las *contraseñas* creadas se pueden asociar a la *categoría "Categoría no especificada"* o a una de las otras trece *categorías* existentes por defecto. Las *categorías* pueden ser modificadas, creadas y eliminadas.
- Cada *contraseña* creada, se compone de los *campos* Correo electrónico, Ingresar, Ingreso secundario, Contraseña, Sitio Web, Nombre, Categoría y Nota, los cuales no son modificables.
- Se tiene un generador de contraseñas, el cual crea aleatoriamente una cadena de acuerdo a la cantidad de caracteres que el usuario desee, además se puede definir si se desean dígitos, letras y símbolos dentro de la cadena.
- Se tiene un menú rápido, con el cual se tiene acceso a todas las *contraseñas*, al generador de contraseñas, a todas las *notas seguras*, a *información personal*, a *medios de pagos*, a *IDs*, al centro para compartir contraseñas, a los contactos de emergencia, entre otros. Al acceder a todas las *contraseñas*, se pueden organizar éstas alfabéticamente o agrupadas por *categoría*, las cuales se ordenan alfabéticamente.

Para complementar la información anterior, se procedió a realizar una encuesta a un grupo de alumnos de tercer año, de la carrera Técnico Universitario en Informática, de la Universidad Técnica Federico Santa María, Sede Viña del Mar. La encuesta aborda temas referentes al uso que se le da a los dispositivos móviles, las prácticas que los encuestados realizan en lo que respecta a administrar sus cuentas y contraseñas, y la opinión que los alumnos tienen frente a aplicaciones como las mencionadas anteriormente.

A continuación, se describen las diversas preguntas contenidas en la encuesta, con sus alternativas correspondientes:

- ¿Cuánto tiempo tengo el celular cerca? Marque solo una opción.
    - No tengo un celular.

- Lo ando trayendo cuando me acuerdo, pero si se me olvida, no me afecta.
- Siempre lo tengo al alcance, o mejor dicho DEBO tenerlo al alcance.

- ¿Para qué uso el celular? Marque solo una opción.
  - Solo llamadas y mensajes.
  - Los anteriores, más algunas aplicaciones básicas, como WhatsApp, Chrome, Facebook, etc.
  - Los anteriores, más un montón de aplicaciones extras, como Uber, Netflix, Waze, Tinder, etc.
- Poseo una cuenta en las siguientes páginas web o servicios seleccionados. Marque todas las que correspondan.
  - Facebook
  - Twitter
  - Instagram
  - Tumblr
  - Reddit
  - Apple (iTunes, AppStore)
  - Spotify
  - Netflix
  - Google (YouTube, Gmail, Google+, Google Drive)
  - Outlook
  - Battle.Net
  - Riot Games
  - Steam
  - Cuenta RUT Banco Estado
  - Cuenta UTFSM (Siga, Aula Virtual)
- Incluyendo las mencionadas anteriormente, ¿Cuántas cuentas poseo en total, aproximadamente? Marque una opción.
  - 10
  - 20
  - 30
  - 40
  - 50 o más
- ¿He sido víctima de un acceso no autorizado, “hackeo”, a alguna de mis cuentas? Marque una opción.
  - Sí, en los últimos dos años.

- Sí, pero hace más de dos años.
- No, nunca he sido víctima.

- ¿Cuál de las siguientes prácticas representa mejor la cantidad de contraseñas distintas que tengo? Marque una opción.
  - Uso una contraseña distinta para cada una de las cuentas que poseo.
  - Uso una única contraseña, repetida en cada una de las cuentas que poseo.
  - No uso una contraseña distinta para cada cuenta, pero tampoco uso una única contraseña para todas las cuentas.
- ¿Cuál de las siguientes prácticas representa mejor la frecuencia con la que cambio mis contraseñas? Marque una opción.
  - Cambio todas mis contraseñas por lo menos una vez al semestre.
  - Cambio todas mis contraseñas por lo menos una vez cada par de años.
  - No recuerdo la última vez que cambie mis contraseñas.
- ¿Cuál de las siguientes prácticas representa mejor la composición de mis contraseñas? Marque una opción.
  - Mis contraseñas se componen de caracteres elegidos aleatoriamente.
  - Mis contraseñas tienen alguna relación conmigo. (Nombre de un pariente, mascota, seudónimo)
  - No son aleatorias, pero tampoco tiene relación conmigo. (qwerty, 123456, password)
- ¿He olvidado alguna vez el nombre de usuario o contraseña de una cuenta? Marque una opción.
  - No, porque tengo muy buena memoria o anoto mis datos en algún lugar.
  - Sí, pero solo ocurre una vez cada año o par de años.
  - Sí, cada tres o cuatro meses debo solicitar la recuperación de datos de una cuenta en específico.
- Teniendo presente que repetir contraseñas en diferentes cuentas, no cambiar las contraseñas regularmente y que las contraseñas estén compuestas por caracteres predecibles, son muy malas prácticas en lo que respecta a seguridad informática. ¿Cómo evalúo mis prácticas de seguridad?
  - Seleccione un número de 1 al 10, siendo el 1 “Muy malas prácticas de seguridad” y 10 “Muy buenas prácticas de seguridad”.

- En el mercado existen una serie de aplicaciones que ayudan a usuarios a recordar sus contraseñas, entre otras cosas. (1Password, LastPass, Enpass) ¿Qué opino al respecto? Marque una opción.
    - Uso una de esas aplicaciones, y la encuentro muy útil.
    - No uso dichas aplicaciones, sabía que existían, y podría llegar a usarlas.
    - No uso dichas aplicaciones, sabía que existían, pero no las encuentro de utilidad.
    - No sabía que existían dichas aplicaciones, pero podría llegar a usarlas.
    - No sabía que existían, y no me intereso en probarlas.

Estos son los resultados obtenidos, al realizar las preguntas, mencionadas anteriormente, a un grupo de 27 alumnos:

- ¿Cuánto tiempo tengo el celular cerca?

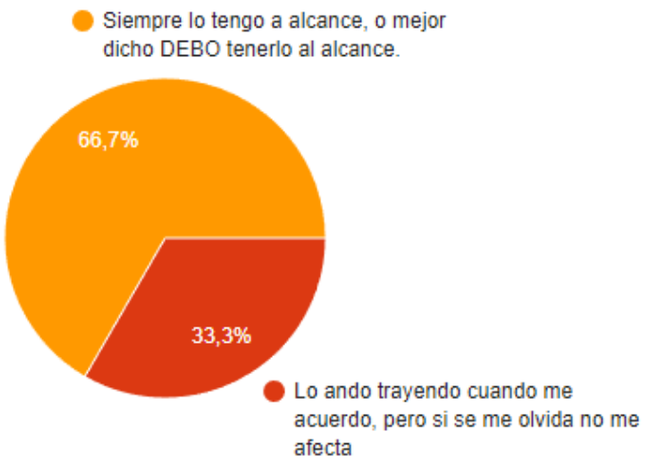

Gráfico 1-1. Resultado pregunta Nº1 de encuesta.

- ¿Para qué uso el celular?

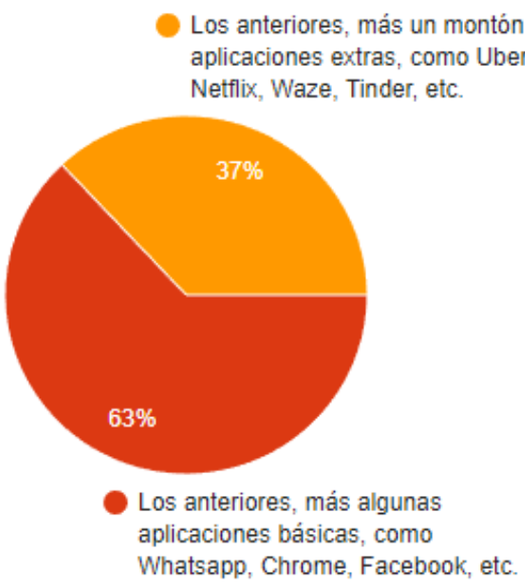

Gráfico 1-2. Resultado pregunta Nº2 de encuesta.

- Poseo una cuenta en las siguientes páginas web o servicios seleccionados.

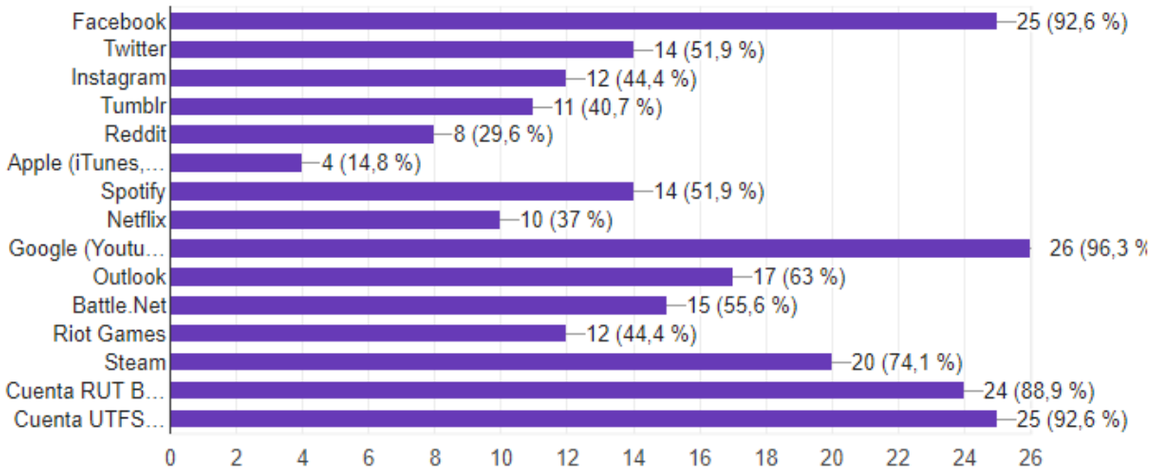

Gráfico 1-3. Resultado pregunta Nº3 de encuesta.

- Incluyendo las mencionadas anteriormente, ¿Cuántas cuentas poseo en total, aproximadamente?

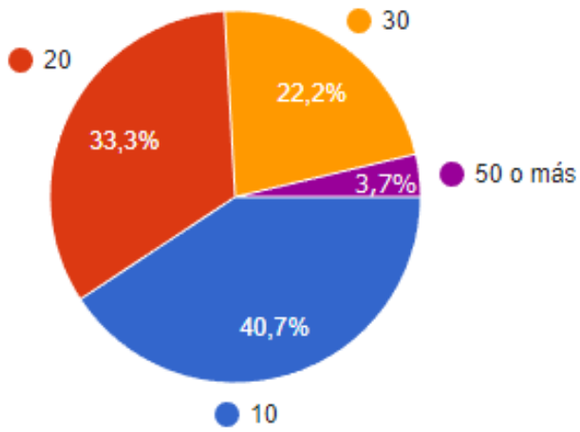

Gráfico 1-4. Resultado pregunta Nº4 de encuesta.

- ¿He sido víctima de un acceso no autorizado, “hackeo”, a alguna de mis cuentas?

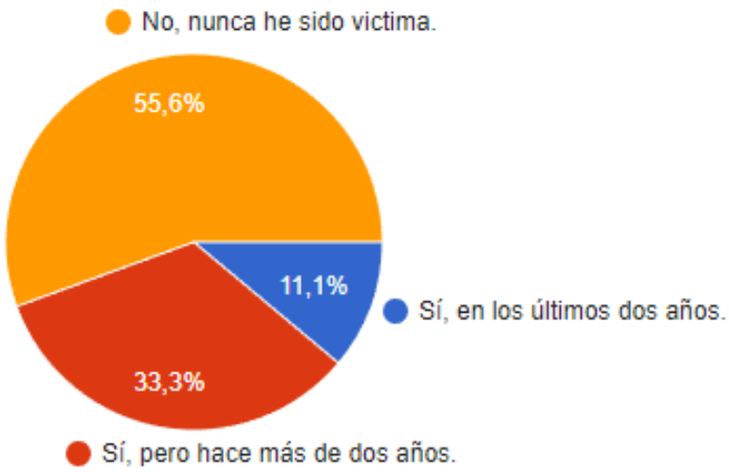

Gráfico 1-5. Resultado pregunta Nº5 de encuesta.

- ¿Cuál de las siguientes prácticas representa mejor la cantidad de contraseñas distintas que tengo?

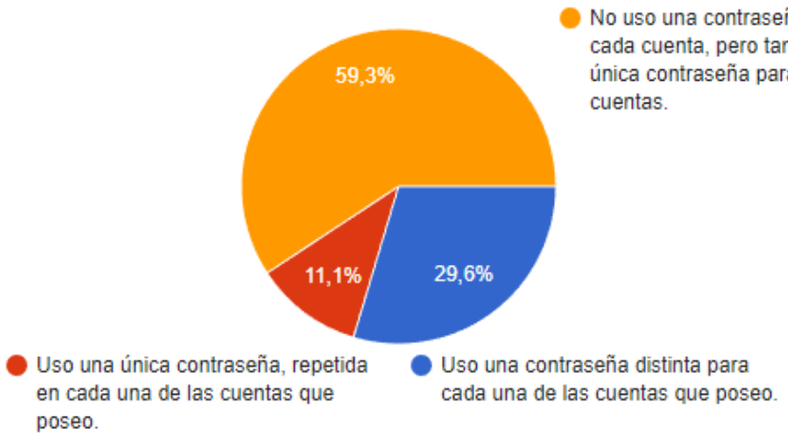

Gráfico 1-6. Resultado pregunta Nº6 de encuesta.

- ¿Cuál de las siguientes prácticas representa mejor la frecuencia con la que cambio mis contraseñas?

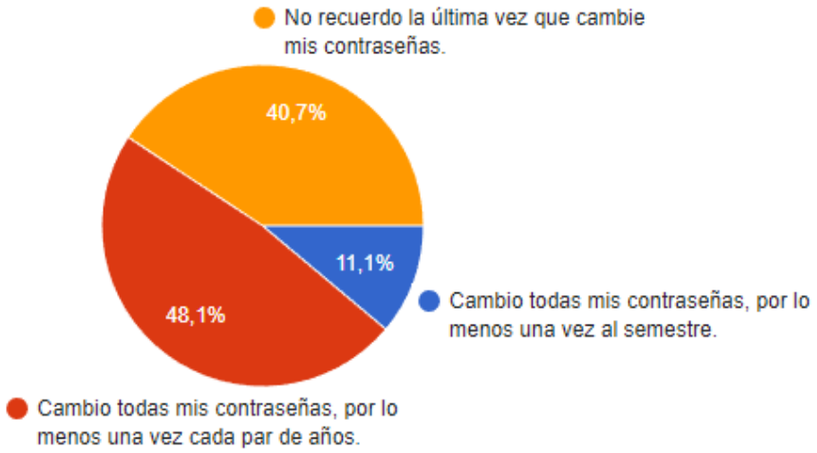

Gráfico 1-7. Resultado pregunta Nº7 de encuesta.

- ¿Cuál de las siguientes prácticas representa mejor la composición de mis contraseñas?

Gráfico 1-8. Resultado pregunta Nº8 de encuesta.

- ¿He olvidado alguna vez el nombre de usuario o contraseña de una cuenta?

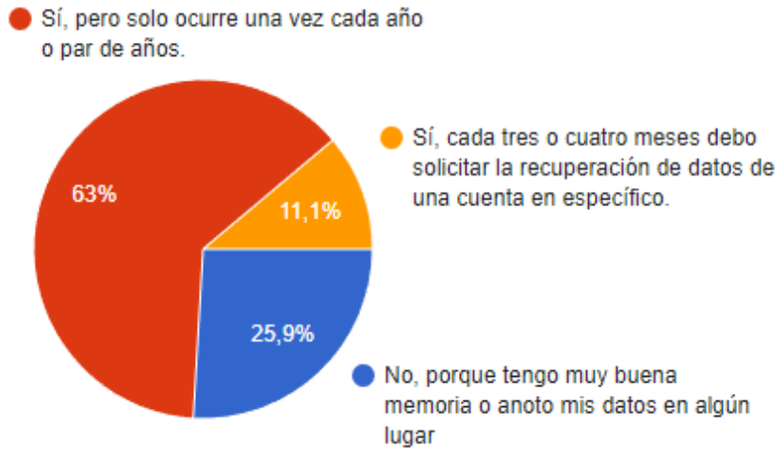

Gráfico 1-9. Resultado pregunta Nº9 de encuesta.

- Teniendo presente que repetir contraseñas en diferentes cuentas, no cambiar las contraseñas regularmente y que las contraseñas estén compuestas por caracteres predecibles, son muy malas prácticas en lo que respecta a seguridad informática. ¿Cómo evalúo mis prácticas de seguridad?

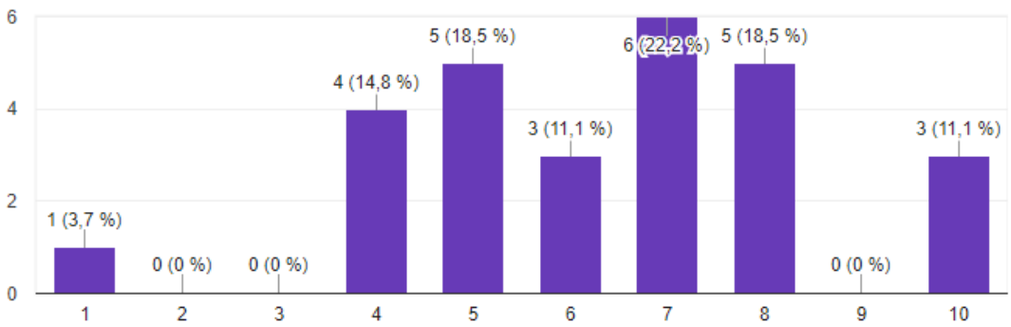

Gráfico 1-10. Resultado pregunta Nº10 de encuesta.

- En el mercado existen una serie de aplicaciones que ayudan a usuarios a recordar sus contraseñas, entre otras cosas. (1Password, LastPass, Enpass) ¿Qué opino al respecto?

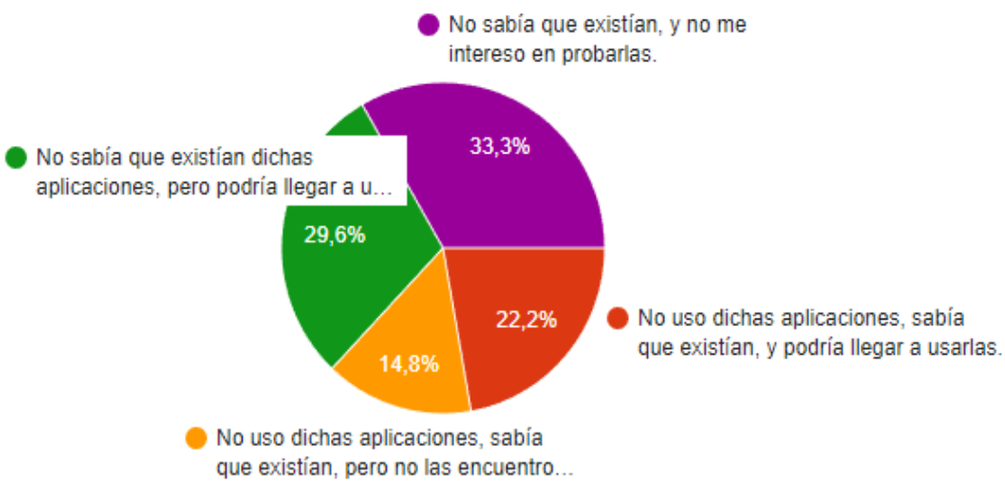

Gráfico 1-11. Resultado pregunta Nº11 de encuesta.

De los resultados obtenidos en la encuesta, se destacan los siguientes aspectos importantes:

- Dos tercios de los encuestados dijeron que deben andar con el teléfono móvil al alcance.
- Todos los encuestados tienen alguna experiencia con el uso de aplicaciones de celulares, ya sea usando aplicaciones básicas (2/3 de los encuestados), o usando aplicaciones mucho más específicas (el restante tercio de los encuestados).
- Los encuestados poseen en promedio 19 cuentas distintas.
- Un poco más de la mitad, nunca ha sido víctima de un acceso no autorizado a una de sus cuentas, mientras que solo una décima parte, ha sufrido dicho ataque en los últimos dos años.
- Casi un 60% de los encuestados repiten contraseñas entre sus distintas cuentas, y un 11% utilizan una contraseña única.
- Un 40% de los encuestados no recuerda cuando cambio sus contraseñas por última vez.
- Una gran minoría, del 11%, posee contraseñas con caracteres elegidos aleatoriamente.
- Solo un 25% de los encuestados no tiene problemas para recordar sus nombres de usuarios o contraseñas.
- Los encuestados evaluaron sus prácticas con un promedio de 6,4 puntos.
- Un 51,8% de los encuestados está abierto a la posibilidad de usar una aplicación de las mencionadas anteriormente.

Es interesante ver cómo un 70% de los encuestados repiten contraseñas en diferentes cuentas, 90% no actualizan frecuentemente sus contraseñas y 50% usan contraseñas relacionadas con ellos mismos, todas consideradas prácticas muy inseguras, pero a pesar de esto, un 50% de los encuestados se evaluó a si mismo con una nota 7 o superior. La teoría detrás de dichos números es que, como un 90% de los encuestados no ha sufrido un acceso no autorizado en los últimos dos años, subestiman la importancia de tener buenas prácticas en lo que respecta a seguridad informática, a esto se le puede agregar que uno, como estudiante, no suele ser un objetivo para el robo de información. Por otro lado, tenemos que, un poco más de la mitad de los encuestados, no eliminan la posibilidad de usar una aplicación para ayudarlos a administrar sus cuentas y contraseñas, lo que indica la existencia de un mercado, que el presente proyecto puede satisfacer.

1.2. PROBLEMAS DETECTADOS

Sin uso de herramientas informáticas:

- Al utilizar una misma contraseña para distintos servicios informáticos, en caso de que uno de los servicios se vea comprometido, el usuario se puede encontrar en la situación de que sus cuentas, de otros servicios, también se encuentren vulneradas.
- Al utilizar cadenas de caracteres predecibles, como lo son los datos asociados al usuario (fecha de nacimiento, dirección, nombre de parientes, entre otros) o como lo son las cadenas típicas utilizadas en contraseñas (“123456”, “qwerty”, entre otros), el usuario se puede encontrar en la situación de que un atacante adivine las contraseñas que éste posee, mediante el conocimiento de los datos mencionados.
- Al utilizar una misma contraseña por largos periodos de tiempo, el usuario corre el riesgo de ser víctima de un ataque por fuerza bruta, el cual es difícil de llevar a cabo si la contraseña es actualizada frecuentemente.

Con uso de 1Password:

- Tiene un costo monetario obligatorio para el usuario.
- No se tiene un sistema para organizar los *ítems* de acuerdo al gusto del usuario, ya que las *categorías* no pueden ser modificadas, y los *tags*, que sí son definidos por el usuario, cumplen una función más similar al de filtrado por palabras claves.
- No es posible definir con exactitud la composición de las contraseñas creadas por el generador automático. Es decir, se puede definir que estén compuestas por símbolos, pero no se puede definir explícitamente que símbolos.
- No es posible definir el tiempo en que se considera una contraseña como obsoleta. El sistema solo informa cuando hay contraseñas cuya vejez esté entre los 6 y 12 meses, entre 1 y 3 años, y más de 3 años.

Con uso de LastPass:

- No se puede asociar un *sitio* a más de un *directorio*.
- No es posible cambiar el orden en que aparecen los *directorios* al ser mostrados, ni el orden en que los *sitios* aparecen dentro de un *directorio*.
- No es posible definir con exactitud la composición de las contraseñas creadas por el generador automático.

- No se posee un sistema para informar de contraseñas obsoletas por el paso del tiempo.

Con uso de Enpass:

- A pesar de poseer una versión gratuita, sus grandes limitaciones terminan por obligar al usuario a tener que pagar un valor monetario.
- No se pueden crear nuevas *categorías*, modificar o eliminar las *categorías* que vienen por defecto.
- El sistema para organizar los *elementos* mediante *carpetas* puede resultar confuso de navegar para usuarios no acostumbrados a ello, además que dicha navegación puede tomar tiempo extra.
- No es posible definir con exactitud la composición de las contraseñas creadas por el generador automático.
- No es posible cambiar de posición los *accesos directos* a las diferentes *carpetas*, teniendo que movilizarse hasta el final del menú rápido para encontrarlos y luego buscar, entre todos ellos, el que uno requiere.
- No se posee un sistema para informar de contraseñas obsoletas por el paso del tiempo.

Con uso de Dashlane:

- No es conveniente utilizar la versión gratuita, por no tener la funcionalidad de respaldo o sincronización, lo que significa un costo monetario obligatorio para el usuario.
- No se puede asociar una *contraseña* a más de una *categoría*.
- No es posible definir con exactitud la composición de las contraseñas creadas por el generador automático.
- No es posible cambiar el orden en que aparecen las *categorías*, ni el orden en que aparecen las *contraseñas* dentro de la *categoría* correspondiente.
- No se posee un sistema para informar de contraseñas obsoletas por el paso del tiempo.

1.3. DESCRIPCIÓN DEL SISTEMA PROPUESTO

1.3.1. OBJETIVOS Y BENEFICIOS DEL SISTEMA PROPUESTO

1.3.1.1. Objetivo Principal

Crear un sistema informático que permita administrar las cuentas de usuario, y el historial de contraseñas asociado a ellas, de manera segura, que permita organizar dichos datos a la medida, que entregue herramientas para generar contraseñas no predecibles y para solicitar el cambio de éstas debido al paso del tiempo.

1.3.1.2. Objetivos Específicos

- Mantener los datos asociados a las cuentas y las contraseñas del usuario.
- Para cada cuenta, mantener un historial de las contraseñas que han estado relacionadas con ésta.
- Mantener una clasificación de cuentas, llamadas categorías, que son administradas por el usuario.
- Encriptar los datos sensibles, mediante metodologías criptográficas.
- Generar contraseñas aleatorias cuando el usuario lo requiera, conformadas por caracteres de su preferencia o palabras aleatorias.
- Informar la necesidad de actualizar la contraseña de determinada cuenta, debido al tiempo que ha pasado desde la última actualización.
- Respaldar la base de datos en la nube, cuando el usuario lo desee y se tenga la conexión pertinente. Y usar dicho respaldo para llenar la base de datos local, al momento de instalar la aplicación en un nuevo móvil.

1.3.1.3. Beneficios

- Cuando el usuario requiera iniciar sesión en un sistema informático, la aplicación ayudará al usuario a recordar la contraseña que tiene asociada a dicha cuenta.

- Cuando el usuario necesite una de las contraseñas antiguas de una cuenta, por ejemplo, cuando se intenta recuperar una cuenta perdida, la aplicación le recordará ésta, mediante el historial de contraseñas.
- Mediante las categorías y el posicionamiento de las cuentas en los menús, la aplicación le otorga al usuario, un rápido acceso a la información que éste requiera.
- Gracias a las metodologías criptográficas utilizadas por la aplicación, solo los usuarios que posean la llave maestra podrán tener acceso a la información sensible del sistema.
- Al otorgar una herramienta que genere contraseñas aleatorias, se le permite al usuario eliminar la predictibilidad de éstas, reduciendo la posibilidad de ataques exitosos.
- Gracias a la capacidad de informar cuando se debe actualizar la contraseña de una cuenta, el usuario sabrá cuándo se deben realizar estos cambios, reduciendo la posibilidad de ataques exitosos.
- Debido al respaldo de los datos que se posee en la nube, es posible recuperar la información de la base de datos, en caso de pérdida del equipo.

1.3.2. DESCRIPCIÓN GENERAL DE LA SOLUCIÓN PROPUESTA

Mediante una pantalla de identificación, se le pedirá al usuario la llave maestra del sistema, la cual será utilizada en la validación del usuario y en la desencriptación de los datos de la base de datos. Una vez el usuario es validado correctamente, se le da acceso al menú principal de la aplicación, con el cual se podrá mover por las diversas pantallas principales del sistema.

Dependiendo de la pantalla principal en la que el usuario se encuentre, se podrán observar las cuentas existentes agrupadas por categorías, las diferentes categorías, los parámetros de configuración y la información acerca de la aplicación.

Desde las pantallas principales, dependiendo de éstas, se podrá ingresar a una nueva serie de pantallas más específicas, donde el usuario podrá observar los detalles asociados a una cuenta, incluyendo su nombre, descripción y contraseña actual asociada, se podrá agregar nuevas cuentas, modificar cuentas existentes, eliminar cuentas no deseadas, observar o eliminar el historial de contraseñas asociado a una cuenta, observar los detalles asociados a una categoría, incluyendo su nombre y las cuentas asociadas a ésta, agregar nuevas categorías,

modificar categorías existentes, eliminar categorías no deseadas, se podrá modificar los valores de los parámetros de configuración de la aplicación, administrar la llave maestra y, por último, respaldar la base de datos en la nube.

El sistema propuesto funcionará en dispositivos móviles, de manera independiente entre ellos, con una base de datos local, y un respaldo en la nube. Se utilizará el paradigma orientado a eventos y orientado a objetos, con una interfaz gráfica para ayudar al usuario en su comprensión.

1.3.3. DIAGRAMA DE FLUJO ADMINISTRATIVO

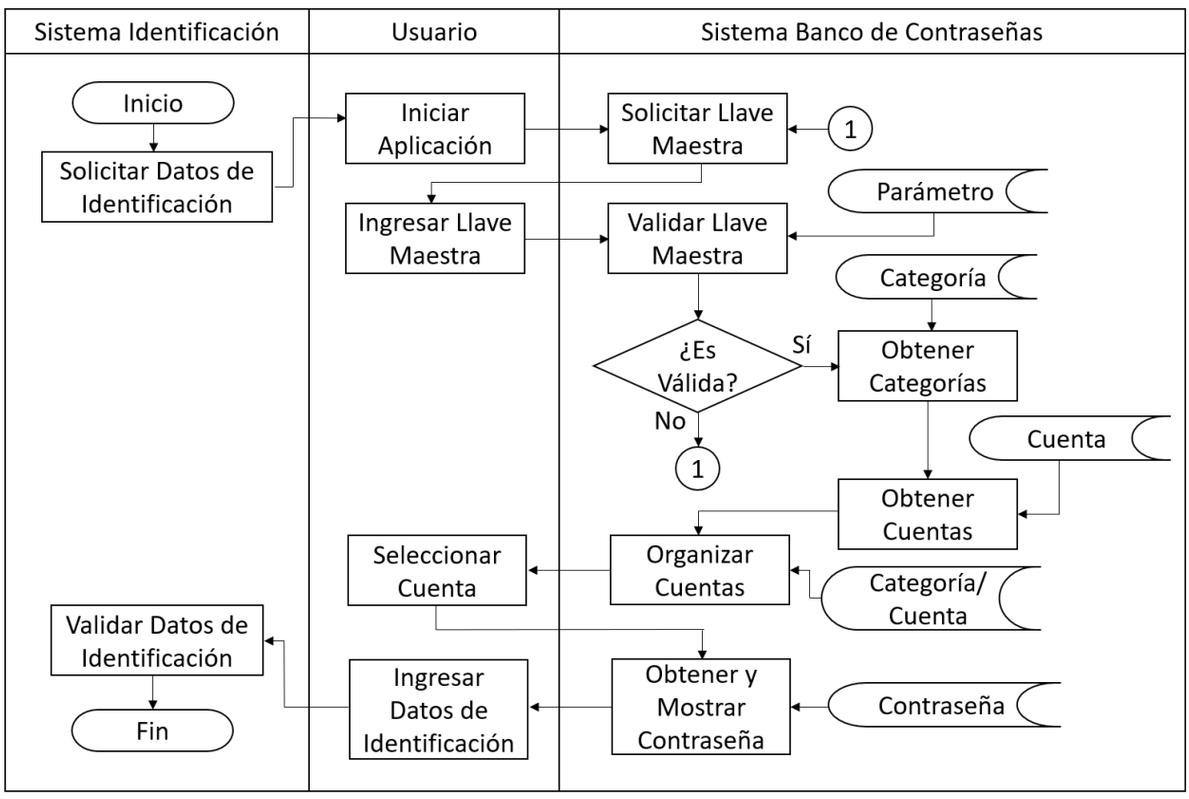

Figura 1-1. Diagrama de Flujo Administrativo.

En el diagrama presentado, cabe ser mencionado que el Sistema de Identificación corresponde a cualquier otro sistema, externo al Sistema del Banco de Contraseñas, que haga solicitud al usuario de datos de autentificación como, por ejemplo, el sistema de autentificación utilizado por servicios de correo electrónico, con una dirección de correo y una contraseña, o el sistema de autentificación de un cajero automático, con la clave de acceso del cliente.

1.3.4. ESTRUCTURA FUNCIONAL DEL SISTEMA

A continuación, se presenta la estructura funcional del sistema:

1. Validar acceso a la aplicación.
   Se pide el ingreso de la llave maestra, para acceder a la aplicación y desencriptar los datos de la base de datos. Para almacenar la llave maestra, de manera segura en la base de datos, se hará uso del algoritmo “PBKDF2WithHmacSHA1”, también cabe mencionar la utilización del algoritmo “SHA1PRNG” para la generación de una SALT, también conocida como SAL, que corresponde a un conjunto de bytes que se combina con la llave maestra para impedir la utilización de ataques mediante tablas arcoíris.
2. Cerrar sesión de la aplicación.
   Se podrá cerrar la sesión de la aplicación, de tal manera que se requiera el ingreso de la llave maestra para su uso posterior.
3. Detallar información de la aplicación.
   Se le entregará información al usuario respecto a detalles de la aplicación, como lo son los nombres de los creadores de ésta, la versión de ésta, nombre de otros participantes, etc.
4. Administrar la llave maestra.
   Se permitirá al usuario crear o cambiar su llave maestra de acceso, modificando la encriptación de todos los datos acordemente.
5. Exportar/Importar la base de datos.
   Mediante una copia de la base de datos encriptada, la cual se encontrará asociada a una cuenta de Google Drive, el usuario podrá respaldar la base de datos local de su dispositivo o podrá completar los datos de ésta, mediante los datos de la base de datos de respaldo. Para lograr lo anterior, se hará uso de la GOOGLE DRIVE API, cuya funcionalidad central consiste en la entrega de librerías que son usadas para la descarga y carga de archivos a Google Drive.
6. Desfragmentar la base de datos.
   Debido al funcionamiento de la asignación de IDs de SQLite, es necesario reorganizar la base de datos después de determinada cantidad de entidades creadas y eliminadas. Esto se llevará a cabo mediante la utilización de los comandos, de agregar, modificar, consultar y eliminar registros, que posee SQLite.

7. Encriptar/Desencriptar campos de la base de datos.

Se encriptará, y desencriptará acordemente, los datos sensibles de la base de datos mediante la utilización de la llave maestra. El algoritmo a utilizar corresponde al "AES/CBC/PKCS5Padding".

8. Informar necesidad de actualizar contraseña.

Se le informará al usuario, mediante iconos especiales en los listados, la necesidad de actualizar la contraseña asociada a determinada cuenta, debido al tiempo que ha pasado desde la última actualización.

9. Generar contraseñas aleatorias.

Cuando el usuario lo requiera, la aplicación podrá generar contraseñas aleatorias para las cuentas que él posee. Se podrá elegir dos modalidades, por caracteres aleatorios, donde uno define la cantidad y composición de los caracteres, y por palabras aleatorias, donde uno define la cantidad de palabras y el conector a usar para unirlas.

10. Crear, modificar, eliminar y consultar categorías.

Se podrá crear, modificar, eliminar y consultar categorías de la base de datos, las cuales serán utilizadas para organizar las cuentas, y se compondrán por un nombre y una posición de ordenamiento.

11. Crear, modificar, eliminar y consultar cuentas.

Se podrá crear, modificar, eliminar y consultar cuentas de la base de datos, las cuales estarán asociadas con diversas categorías, y se componen de un nombre, descripción y tiempo de validez.

12. Crear, modificar, eliminar y consultar categorías/cuentas.

Se podrá crear, modificar, eliminar y consultar las relaciones entre categorías y cuentas, las cuales se componen del nombre de la categoría, el nombre de la cuenta, y la posición de ordenamiento que tendrá dicha cuenta, dentro de dicha categoría.

13. Crear, eliminar y consultar contraseñas.

Se podrá crear, eliminar y consultar contraseñas de la base de datos, las cuales estarán asociadas a una cuenta, y se compondrán de un ID, un valor, y una fecha.

14. Modificar y consultar parámetros.

Se podrá modificar y consultar parámetros de la base de datos, los cuales serán usados para configurar la aplicación, y se compondrán de un nombre, un valor y una posición de ordenamiento.

15. Generador de consultas de cuentas.

Genera una consulta de una cuenta en específico.

16. Generador de consultas de contraseña.
Genera una consulta de la última contraseña asociada a una cuenta.

17. Generador de lista de categorías.
Genera una lista de todas las categorías en existencia.

18. Generador de lista de cuentas.
Genera una lista de las cuentas asociadas a una categoría.

19. Generador de lista de contraseñas.
Genera una lista de las contraseñas asociadas a una cuenta.

20. Generador de lista de parámetros.
Genera una lista de los parámetros que son visible para el usuario.

1.3.5. INFORMACIÓN A MANEJAR

1.3.5.1. Salidas del Sistema

- Consulta de cuenta.
  Se mostrarán los datos de una cuenta en específico, tales como su nombre, descripción y tiempo de validez.
- Consulta de contraseña.
  Se mostrarán los datos de la última contraseña asociada a una cuenta en específico, tales como el valor de ésta.
- Lista de categorías.
  Se mostrará un listado de todas las categorías en existencia, ordenadas de acuerdo con su campo posición, y entregando datos tales como el nombre de ésta.
- Lista de cuentas.
  Se mostrará un listado de todas las cuentas asociadas a una categoría en específico, ordenadas de acuerdo a su campo posición, y entregando datos tales como el nombre y descripción de ésta.
- Lista de contraseñas.
  Se mostrará un listado de todas las contraseñas asociadas a una cuenta en específico, ordenadas de acuerdo a su campo de ID, y entregando datos tales como el valor y la fecha de creación de ésta.

- Lista de parámetros.

  Se mostrará un listado de todos los parámetros cuya posición sea distinta a nulo, ordenados por ese mismo campo, y entregando datos tales como el nombre y valor de éste.

1.3.5.2. Entradas del Sistema

- Datos de categoría.

  Datos tales como el nombre de la categoría y la posición que tendrá al ser ordenada.
- Datos de cuentas.

  Datos tales como el nombre de la cuenta, la descripción y el tiempo de validez de las contraseñas asociadas.
- Datos de categoría/cuenta.

  Datos tales como el nombre de la categoría y el nombre de la cuenta que se relacionan, además de la posición que tendrá dicha cuenta al ser ordenada dentro de dicha categoría.
- Datos de contraseñas.

  Datos tales como el valor de una contraseña, la fecha en que fue creada y el nombre de la cuenta a la que está asociada.
- Datos de parámetros.

  Datos tales como el valor de un parámetro de configuración.
- Datos de la base de datos de respaldo.

  Cuando se requiera cargar la base de datos de respaldo en la base de datos local del dispositivo, se hará lectura de toda la composición del archivo correspondiente al respaldo, y se copiará dichos datos en el archivo correspondiente a la base de datos local del sistema.

1.3.5.3. Entidades de Información

- Categoría

  Contiene información respecto a las diversas categorías, en las cuales se podrán agrupar las diversas cuentas creadas por el usuario, incluyendo el nombre y la posición en la que se encuentra al momento de ser ordenadas.

- Cuenta

  Contiene información respecto a las diversas cuentas que el usuario vaya creando, como lo son el nombre, la descripción y el tiempo de validez que tendrá la contraseña asociada.

- Categoría/Cuenta

  Corresponde a una entidad de intersección, la cual contiene información respecto a las relaciones entre categorías y cuentas, como lo son el nombre de la categoría, el nombre de la cuenta y la posición que tendrá dicha cuenta al ser mostrada como integrante de dicha categoría.

- Contraseña

  Contiene información respecto a las diversas contraseñas que el usuario vaya creando para cada una de las cuentas existentes, como lo son el nombre de la cuenta a la que está asociada, la fecha en que se creó y el valor que el usuario o el generador automático le asignó.

- Parámetro

  Contiene información respecto a los parámetros de la aplicación, utilizados para la configuración de ésta, como lo son el nombre del parámetro, el valor asociado y la posición que tendrá al ser mostrado, si no se quiere mostrar una entrada determinada, tendrá un valor nulo en la posición. Por ejemplo, el largo de las contraseñas generadas por la aplicación tendrá una entrada en la tabla de parámetros.

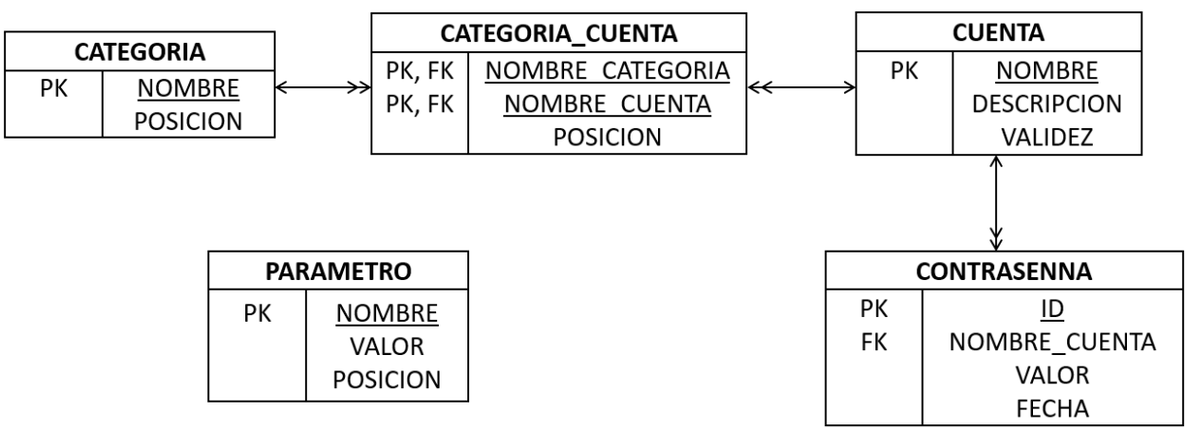

Figura 1-2. Modelo Lógico del Sistema.

Además, se tendrá una segunda base de datos, llamada “diccionario”, la cual solo contendrá una tabla, y será utilizada por el generador de contraseñas aleatorias cuando esté en modalidad de palabras, dicha tabla solo permitirá consultas.

- Palabra

  Contiene información respecto a todas las posibles palabras que pueden ser utilizadas por el generador de contraseñas, tal como el identificador de la palabra y el valor de la palabra en sí.

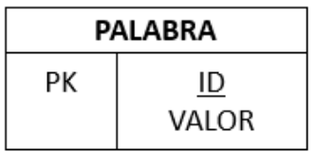

Figura 1-3. Modelo Lógico de la Base de Datos “diccionario”.

1.3.6. ESTRUCTURA DE CÓDIGOS

El campo ID, que corresponde a la clave primaria de la entidad Contraseña, tendrá una estructura numérica correlativa incremental. Cabe destacar la importancia de conservar el orden de dicho campo, ya que no solo se utilizará para identificar una entrada, sino que también será utilizada para identificar el orden en que fueron creadas las diversas contraseñas.

1.3.7. CONDICIONANTES DE DISEÑO

La aplicación será ejecutada en dispositivos móviles, los cuales deberán poseer un sistema operativo Android, versión 4.1 o superior. Será programada con la utilización del entorno de desarrollo integrado Android Studio, el cual utiliza a Java como lenguaje de programación principal. Además, se hará utilización del sistema de gestión de base de datos SQLite para la manipulación de los datos que utilice la aplicación.

CAPÍTULO 2: MEDIO AMBIENTE COMPUTACIONAL Y DESCRIPCIÓN DE ARCHIVOS

2. MEDIO AMBIENTE COMPUTACIONAL Y DESCRIPCIÓN DE ARCHIVOS

2.1. CARACTERÍSTICAS DEL RECURSO COMPUTACIONAL

Para describir los recursos computacionales que poseerá el sistema a desarrollar, ya que el sistema se implementará en múltiples dispositivos smartphones, se considerará el equipo con características más limitadas, que tenga un sistema operativo Android 4.1, y que se pudo conseguir de manera física, es decir, el LG Optimus L7X P714.

2.1.1. CONFIGURACIÓN DEL SISTEMA

2.1.1.1. Tipo de Procesador

Se cuenta con un procesador Dual-Core 1 GHz Cortex-A5, diseñado por ARM Holdings, con un Chipset Qualcomm MSM8225 Snapdragon, y una GPU Adreno 203.

2.1.1.2. Tipo y Capacidad de Almacenamiento, y Mecanismos de Respaldo

Se cuenta, como almacenamiento primario, una memoria RAM de 768 MB, Single-Channel, LPDDR2, 300 MHz, como almacenamiento secundario, se cuenta con una memoria interna de 4GB, expandible hasta 32 GB con microSD, y como mecanismos de respaldo, no se tiene implementado alguno por defecto, pero la aplicación a desarrollar implementará su propio mecanismo para respaldar información, mediante una conexión a la nube de Google Drive.

2.1.1.3. Tipo y Número de Pantallas e Impresoras

El dispositivo cuenta con una única pantalla LCD IPS capacitiva, con diseño Full Touch Screen, 16M colores, con una densidad de píxeles de 217 ppi, una resolución de 480 x 800 píxeles, y un tamaño de 4,3 pulgadas. No se cuenta con algún dispositivo de impresión conectado al smartphone, y no se conocen de métodos que permitan ampliar la cantidad de pantallas asociadas al dispositivo.

2.1.2. SOFTWARE UTILIZADO

2.1.2.1. Sistema Operativo

La elección del sistema operativo, Android versión 4.1, se realizó teniendo en conocimiento que el 98,4% de dispositivos que usan la plataforma Android, tienen la versión mencionada o una versión superior.

Tabla 2-1. Cantidad relativa de dispositivos que usan Android, según versión.

Versión	Nombre	API	Distribución
2.3.3 – 2.3.7	Gingerbread	10	0,8%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	0,8%
4.1.X	Jelly Bean	16	3,1%
4.2.X		17	4,4%
4.3		18	1,3%
4.4	KitKat	19	18,1%
5.0	Lollipop	21	8,2%
5.1		22	22,6%
6.0	Marshmallow	23	31,2%
7.0	Nougat	24	8,9%
7.1		25	0,6%
Distribución Total Versión 4.1 y superior.			**98,4%**

Fuente: developer.android.com, “Paneles de control”, 05/06/2017.

Cabe mencionar que el sistema operativo Android está basado en el núcleo Linux, fue desarrollado inicialmente por Android Inc., y comprado posteriormente por Google, y el primer móvil que se vendió, con el sistema operativo, fue el HTC Dream en octubre del 2008. Para finales del 2016, Android poseía el 81,7% de la cuota de mercado de sistemas operativos para smartphones.

Tabla 2-2. Cuota de Mercado de Sistemas Operativos para Smartphones, 4Q16.

Sistema Operativo	4º Trimestre 2016 Cuota de Mercado (%)
Android	81,7%
Ios	17,9%
Windows	0,3%
BlackBerry	0,0%
Otros OS	0,1%

Fuente: www.gartner.com, 15/02/2017.

Entre las características que se destacan del sistema operativo, se encuentra el hecho de que Android es un proyecto de fuente abierta, “open source project”, es decir, cualquier fabricante de hardware puede construir un dispositivo que ejecute el sistema operativo Android, además, cada versión nueva de Android provee las compatibilidades necesarias para que las aplicaciones desarrolladas, para versiones anteriores, puedan ser ejecutadas sin problemas en versiones posteriores.

2.1.2.2. Herramientas de Desarrollo de Software

La herramienta principal, en lo que respecta al desarrollo de software, corresponde a Android Studio, entorno de desarrollo integrado oficial para la plataforma Android, el cual fue lanzado en diciembre del 2014, y reemplazó a Eclipse como IDE oficial. Cabe destacar que Android Studio fue programado en Java, y utiliza dicho lenguaje como lenguaje de programación principal, además hay que mencionar que utiliza el lenguaje XML para lo referente a interfaz gráfica.

En lo que respecta al sistema de gestión de bases de datos relacionales, se tiene al motor SQLite, el cual viene, por omisión, implementado en los sistemas operativos Android.

2.2. DESCRIPCIÓN DE ARCHIVOS

2.2.1. DESCRIPCIÓN GENERAL DE ARCHIVOS

2.2.1.1. Categoría

La tabla Categoría, se encargará de almacenar información asociada a cada categoría, tal como el nombre y la posición que tendrá al ser listada. Las categorías serán usadas para clasificar las cuentas.

2.2.1.2. Cuenta

La tabla Cuenta, se encargará de almacenar información asociada a cada cuenta, tal como el nombre, descripción y tiempo de validez de la contraseña asociada.

2.2.1.3. Categoría/Cuenta

La tabla Categoría/Cuenta, se encargará de almacenar información asociada a la relación entre una categoría y una cuenta, tal como el nombre de la categoría, el nombre de la cuenta y la posición que tendrá dicha cuenta al ser listada como parte de dicha categoría.

2.2.1.4. Contraseña

La tabla Contraseña, se encargará de almacenar información asociada a cada contraseña, tal como el ID, el nombre de la cuenta asociada, el valor y la fecha de creación.

2.2.1.5. Parámetro

La tabla Parámetro, se encargará de almacenar información asociada a los parámetros de configuración de la aplicación, tales como el nombre del parámetro, el valor, la posición que tendrá al ser listado. Esta tabla, entre otros, contendrá lo necesario para validar la llave maestra del usuario.

2.2.2. CODIFICACIÓN DE ARCHIVOS

El único elemento que utiliza alguna codificación especial corresponde a la columna ID, de la tabla Contraseña, el cual consiste en una estructura numérica correlativa incremental.

También cabe destacar que el valor de las contraseñas, es decir la columna VALOR, de la tabla Contraseña, corresponde al valor encriptado de ésta, no al valor como texto plano. Para lo cual se utilizará el algoritmo de encriptación AES, en modalidad CBC, es decir, se requerirá de un vector de inicialización junto a cada encriptación y desencriptación, el cual se almacenará en la misma columna. En otras palabras, el campo VALOR se compondrá de una primera parte, correspondiente al vector de inicialización utilizado para generar dicha encriptación, y una segunda parte correspondiente al valor encriptado de la contraseña.

Por otro lado, tenemos que dentro de la tabla Parámetro siempre deberán existir tres registros, utilizados para la autentificación del usuario, y la encriptación y desencriptación de los datos importantes. Estos registros corresponden a la Sal de Acceso, Hash de Acceso y Sal de Encriptación, los dos primeros son utilizados, junto al algoritmo PBKDF2WithHmacSHA1, para validar la llave maestra que ingresa el usuario al momento de autentificarse, y el último es utilizado, junto a la llave maestra, para generar la llave de encriptación, utilizando el mismo algoritmo mencionado.

2.2.3. DESCRIPCIÓN ESPECÍFICA DE ARCHIVOS

2.2.3.1. Categoría

Tabla 2-3. Descripción de la tabla Categoría.

Nombre Físico	CATEGORIA
Descripción	Almacena todos los datos de las categorías.
Clave Primaria	NOMBRE
Claves Foráneas	No tiene
Longitud de Registro	25

Descripción de Registros		
NOMBRE	Tipo	VARCHAR
	Longitud	20
	Descripción	Nombre con el que se identificará una categoría.
POSICION	Tipo	SMALLINT UNSIGNED
	Longitud	5
	Descripción	Define la posición que tendrá dicha categoría al ser listada.

2.2.3.2. Cuenta

Tabla 2-4. Descripción de la tabla Cuenta.

Nombre Físico	CUENTA	
Descripción	Almacena todos los datos de las cuentas.	
Clave Primaria	NOMBRE	
Claves Foráneas	No tiene	
Longitud de Registro	280	
Descripción de Registros		
NOMBRE	Tipo	VARCHAR
	Longitud	20
	Descripción	Nombre con el que se identificará una cuenta.
DESCRIPCION	Tipo	VARCHAR
	Longitud	255
	Descripción	Descripción general de la cuenta.
VALIDEZ	Tipo	SMALLINT UNSIGNED
	Longitud	5
	Descripción	Define la cantidad de días que deberán pasar para que la contraseña asociada a la cuenta, se consideré como caducada.

2.2.3.3. Categoría/Cuenta

Tabla 2-5. Descripción de la tabla Categoría/Cuenta.

Nombre Físico	CATEGORIA_CUENTA	
Descripción	Almacena todos los datos asociados a la relación entre una categoría y una cuenta.	
Clave Primaria	NOMBRE_CATEGORIA + NOMBRE_CUENTA	
Claves Foráneas	NOMBRE_CATEGORIA (Hacia tabla CATEGORIA) NOMBRE_CUENTA (Hacia tabla CUENTA)	
Longitud de Registro	45	
Descripción de Registros		
NOMBRE_CATEGORIA	Tipo	VARCHAR
	Longitud	20
	Descripción	Nombre con el que se identificará la categoría de la relación
NOMBRE_CUENTA	Tipo	VARCHAR
	Longitud	20
	Descripción	Nombre con el que se identificará la cuenta de la relación.
POSICION	Tipo	SMALLINT UNSIGNED
	Longitud	5
	Descripción	Define la posición que tendrá dicha cuenta al ser listada como elemento de dicha categoría.

2.2.3.4. Contraseña

Tabla 2-6. Descripción de la tabla Contraseña.

Nombre Físico	CONTRASENNA
Descripción	Almacena todos los datos asociados a las contraseñas.
Clave Primaria	ID
Claves Foráneas	NOMBRE_CUENTA (Hacia tabla CUENTA)
Longitud de Registro	295

Descripción de Registros		
ID	Tipo	INTEGER UNSIGNED
	Longitud	10
	Descripción	ID con el que se identificará una contraseña.
NOMBRE_CUENTA	Tipo	VARCHAR
	Longitud	20
	Descripción	Nombre con el que se identificará la cuenta asociada.
VALOR	Tipo	VARCHAR
	Longitud	255
	Descripción	Define el valor encriptado de la contraseña.
FECHA	Tipo	VARCHAR
	Longitud	10
	Descripción	Define la fecha en que fue creada dicha contraseña.

2.2.3.5. Parámetro

Tabla 2-7. Descripción de la tabla Parámetro.

Nombre Físico	PARAMETRO
Descripción	Almacena todos los datos asociados a los parámetros de configuración.
Clave Primaria	NOMBRE
Claves Foráneas	No tiene
Longitud de Registro	288
Descripción de Registros	
NOMBRE	Tipo	VARCHAR
	Longitud	30
	Descripción	Nombre con el que se identificará el parámetro.
VALOR	Tipo	VARCHAR
	Longitud	255
	Descripción	Valor que tendrá asociado dicho parámetro.

POSICION	Tipo	TINYINT UNSIGNED
	Longitud	3
	Descripción	Define la posición que tendrá dicho parámetro al ser listado, o indicará si no será listado.

2.3. DIAGRAMAS UML

2.3.1. DESCRIPCIÓN DE CASOS DE USO

A continuación, se procederá a describir tres casos de uso fundamentales para el desarrollo de la aplicación, los cuales fueron elegidos, como ejemplos, con el objetivo de dar una idea clara de cómo se conformaría el resto de los casos de usos del proyecto a desarrollar.

2.3.1.1. Caso de Uso: Crear Cuenta

Tabla 2-8. Descripción del caso de uso Crear_Cuenta.

Nombre CU	Crear_Cuenta
Actores Involucrados	Cliente y Mantenedor
Precondiciones	El Mantenedor se ha autentificado.
Escenario de éxito	
El cliente solicita una cuenta y el mantenedor inicia la pantalla para agregar cuentas en el sistema. Se invoca el CU Consultar_Categorías para obtener un listado de las categorías a las que puede pertenecer la nueva cuenta. Se ingresan los datos de la nueva cuenta, es decir, el nombre, la descripción, la validez, la contraseña, y se seleccionan las categorías deseadas. Se invoca el CU Consultar_Cuentas para corroborar que el nombre seleccionado no está siendo usado por otra cuenta. Se registra la cuenta nueva. Se invoca al CU Crear_Contraseña, para registrar la contraseña que el usuario seleccionó. Se invoca al CU Crear_Categoría_Cuenta, para registrar las relaciones surgidas entre las categorías existentes y la nueva cuenta. Se informa al cliente que su cuenta fue creada exitosamente.	

Escenario alternativo	
Si el nombre de la cuenta ya existe, y el usuario así lo desea, se invocará al CU Modificar_Cuenta, en el cual se podrán cambiar los datos de una cuenta ya existente.	
Postcondición	La cuenta nueva ha sido registrada.

2.3.1.2. Caso de Uso: Cambiar Llave Maestra

Tabla 2-9. Descripción del caso de uso Cambiar_Llave_Maestra.

Nombre CU	Cambiar_Llave_Maestra
Actores Involucrados	Cliente y Mantenedor
Precondiciones	El Mantenedor se ha autentificado.
Escenario de éxito	
El cliente solicita cambiar su llave maestra y el mantenedor inicia la pantalla para el cambio de la llave maestra. Se invoca al CU Consultar_Parámetros, para corroborar que ya existe una llave maestra. Se ingresa dos veces la nueva llave maestra deseada por el cliente, se valida que sean iguales y cumplan los mínimos requerimientos de seguridad. Se invoca al CU Consultar_Contraseñas, para obtener un listado de todas las contraseñas que deberán actualizar su encriptación. Se invoca al CU Actualizar_Contraseñas, para desencriptar los valores antiguos de contraseñas, usando la llave maestra anterior, volver a encriptarlos, usando la nueva llave maestra elegida por el usuario, y registrar los cambios. Se invoca al CU Actualizar_Parámetros_Acceso, para sobrescribir los valores usados en la autentificación de usuarios, con nuevos valores acordes a la nueva llave maestra. Se informa al cliente que su nueva llave maestra fue registrada exitosamente.	
Escenario alternativo	
Si no existen los valores asociados a una llave maestra anterior, se invocará al CU Crear_Llave_Maestra, el cual creará una nueva llave maestra y reestablecerá la aplicación a un estado por omisión. Si las nuevas llaves maestras no coinciden o no cumplen con los requisitos mínimos de seguridad, se informará de dicho suceso, y se solicitará el reingreso de la nueva llave maestra.	
Postcondición	Se actualizó la llave maestra.

2.3.1.3. Caso de Uso: Eliminar Categoría

Tabla 2-10. Descripción del caso de uso Eliminar_Categoría.

Nombre CU	Eliminar_Categoría
Actores Involucrados	Cliente y Mantenedor

Precondiciones	El Mantenedor se ha autentificado.
Escenario de éxito	
El cliente solicita la eliminación de una categoría existente y el mantenedor inicia la eliminación deseada. Se invoca al CU Consultar_Categoría, para corroborar que existe la categoría que se desea eliminar. Se espera que se confirme la decisión de eliminar la categoría especificada. Si se recibe la confirmación, se procede a registrar la eliminación de la categoría, lo cual se traduce también en la eliminación de las relaciones Categoría/Cuenta correspondientes. Se invoca al CU Actualizar_Categorías, para modificar el campo POSITION de las categorías que no fueron eliminadas, reutilizando la posición que la categoría eliminada dejó disponible. Se informa al cliente que la categoría seleccionada fue eliminada exitosamente.	
Escenario alternativo	
Si la categoría seleccionada para la eliminación no existe, se informa del error, y se cancela la eliminación. Si no se recibe la confirmación de que se desea eliminar la categoría, se cancela la eliminación.	
Postcondición	Se eliminó la categoría deseada.

2.3.2. DIAGRAMAS DE CASOS DE USO

2.3.2.1. Diagrama Caso de Uso: Crear Cuenta

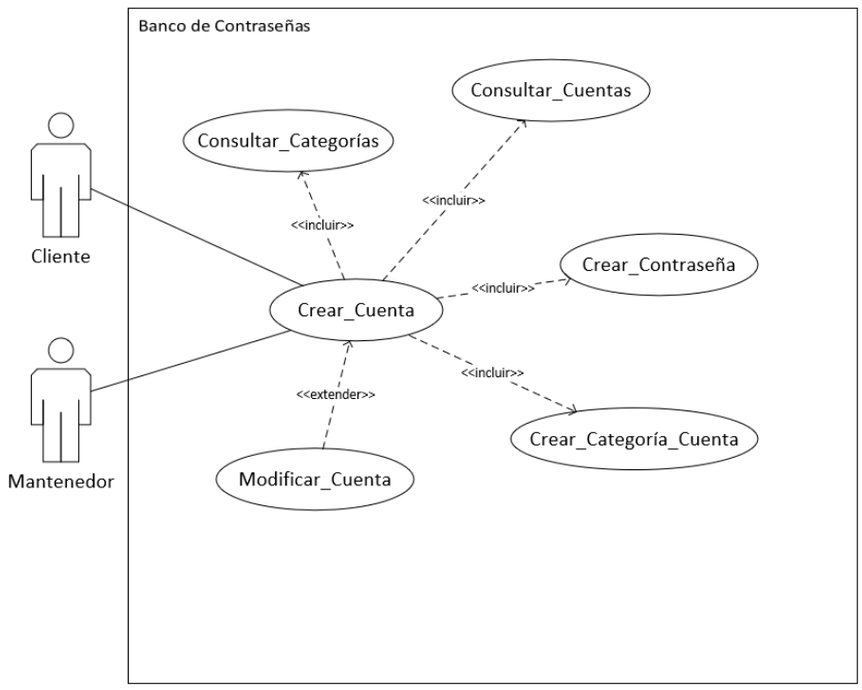

Figura 2-1. Diagrama de caso de uso de Crear_Cuenta.

2.3.2.2. Diagrama Caso de Uso: Cambiar Llave Maestra

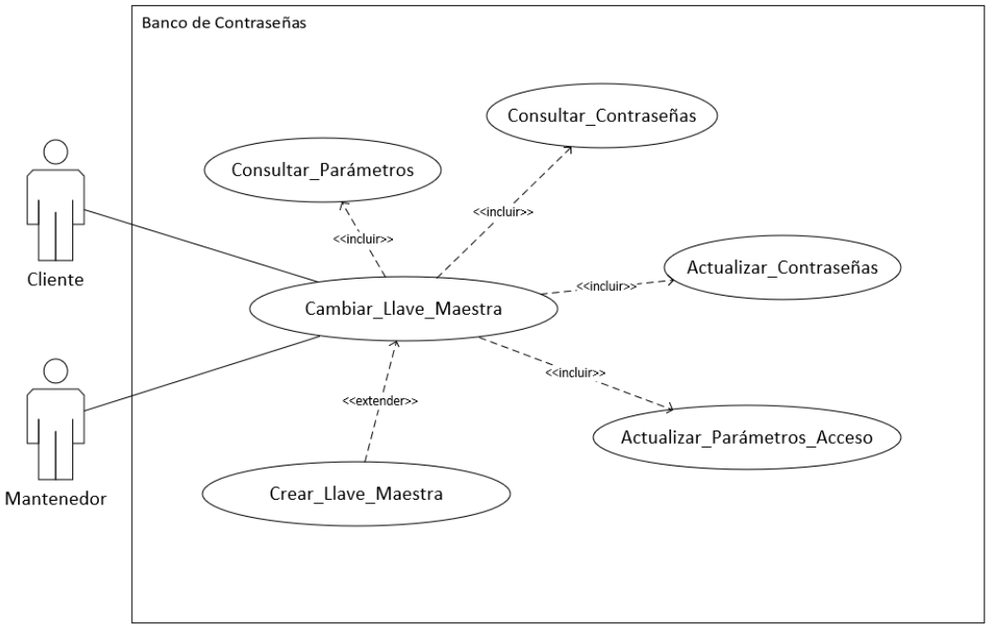

Figura 2-2. Diagrama de caso de uso de Cambiar_Llave_Maestra.

2.3.2.3. Diagrama Caso de Uso: Eliminar Categoría

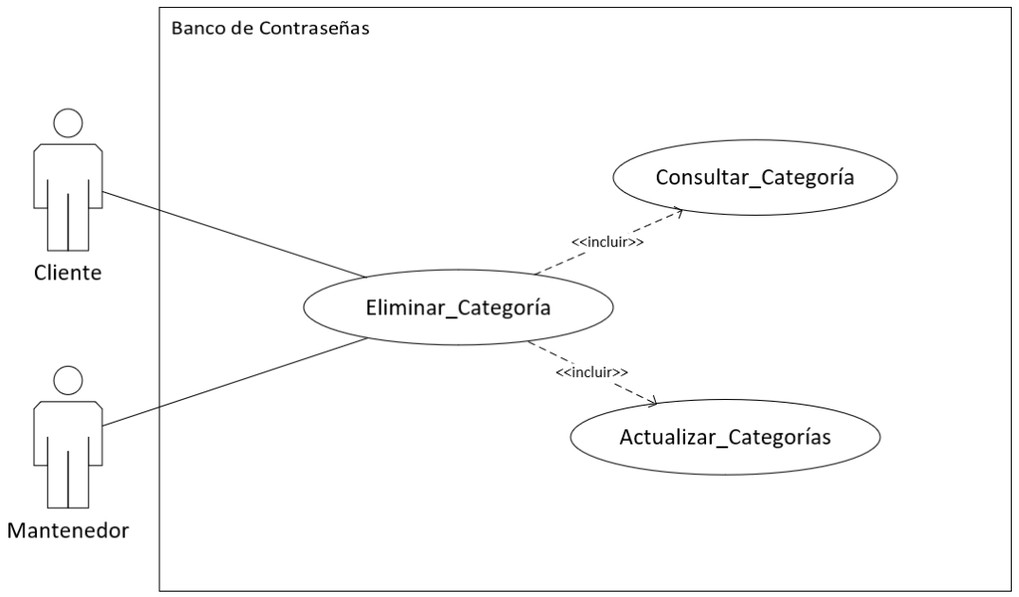

Figura 2-3. Diagrama de caso de uso de Eliminar_Categoría.

2.3.3. DIAGRAMAS DE SECUENCIA

Antes de presentar los diagramas de secuencia, cabe ser mencionado la existencia del objeto DBOperador, el cual cumple una función similar a los DAOs observados en Java EE, es decir, presta al proyecto todas las funcionalidades de los diversos mantenedores. Dicho objeto retornará una instanciación de la clase android.database.Cursor, la cual contendrá toda la información retornada por la sentencia SQL ejecutada.

2.3.3.1. Diagrama Secuencia: Crear Cuenta

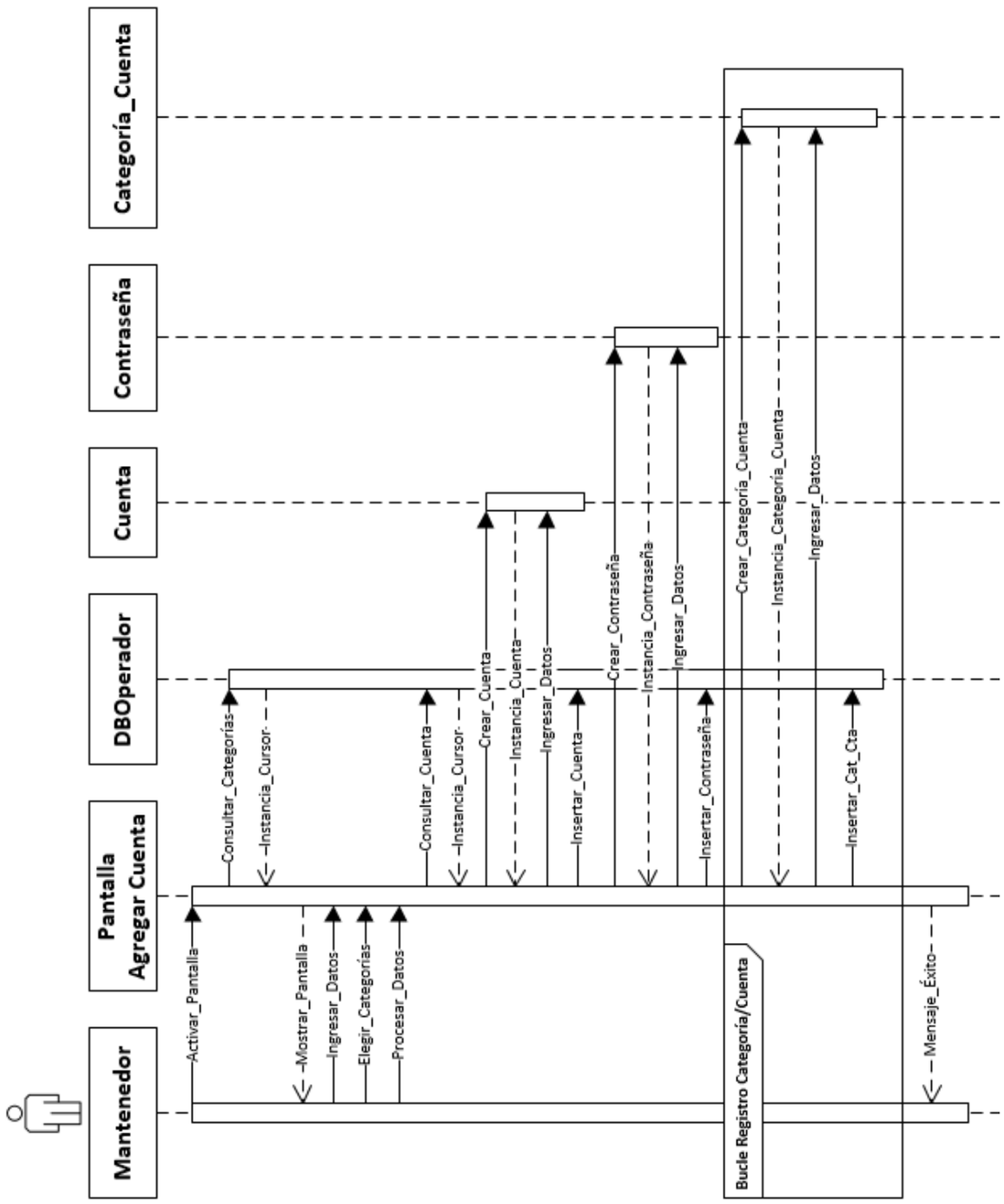

Figura 2-4. Diagrama de secuencia de Crear_Cuenta.

2.3.3.2. Diagrama Secuencia: Cambiar Llave Maestra

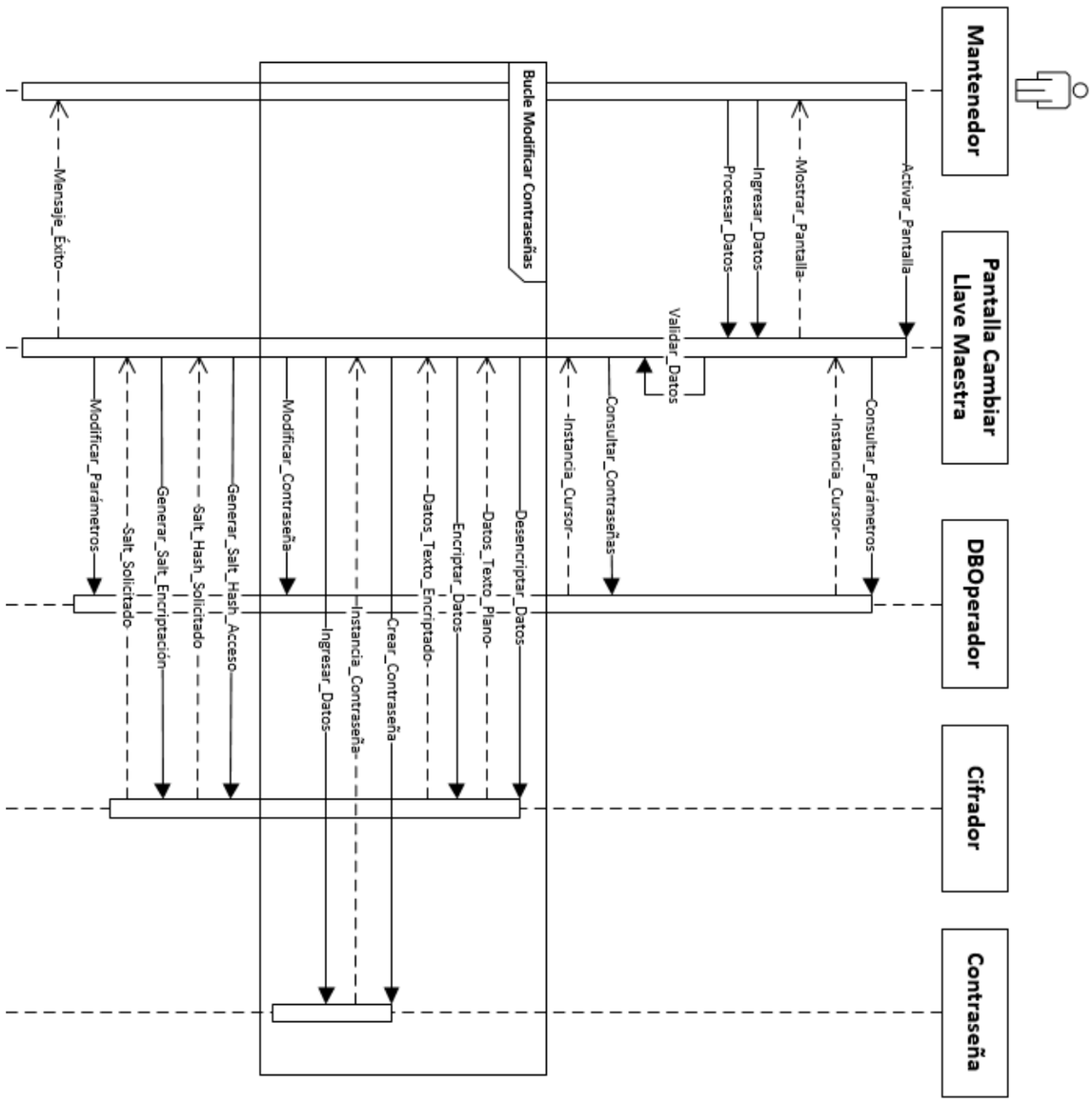

Figura 2-5. Diagrama de secuencia de Cambiar_Llave_Maestra.

2.3.3.3. Diagrama Secuencia: Eliminar Categoría

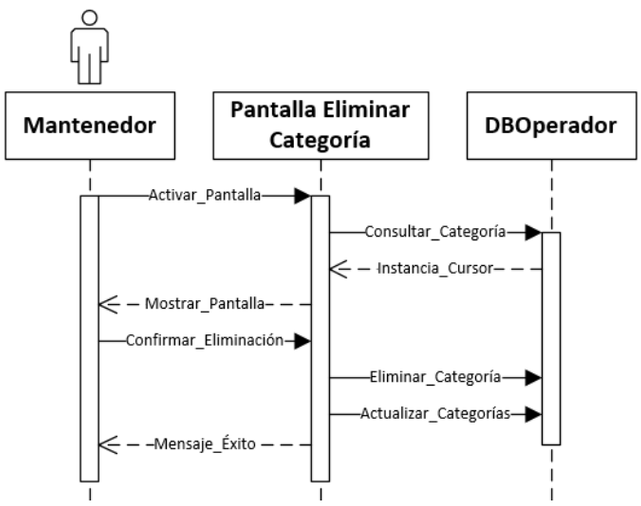

Figura 2-6. Diagrama de secuencia de Eliminar_Categoría.

CAPÍTULO 3: ESTRUCTURA GENERAL DEL SISTEMA

3. ESTRUCTURA GENERAL DEL SISTEMA

3.1. DIAGRAMA MODULAR

A continuación, se muestra el diagrama modular que presenta el sistema:

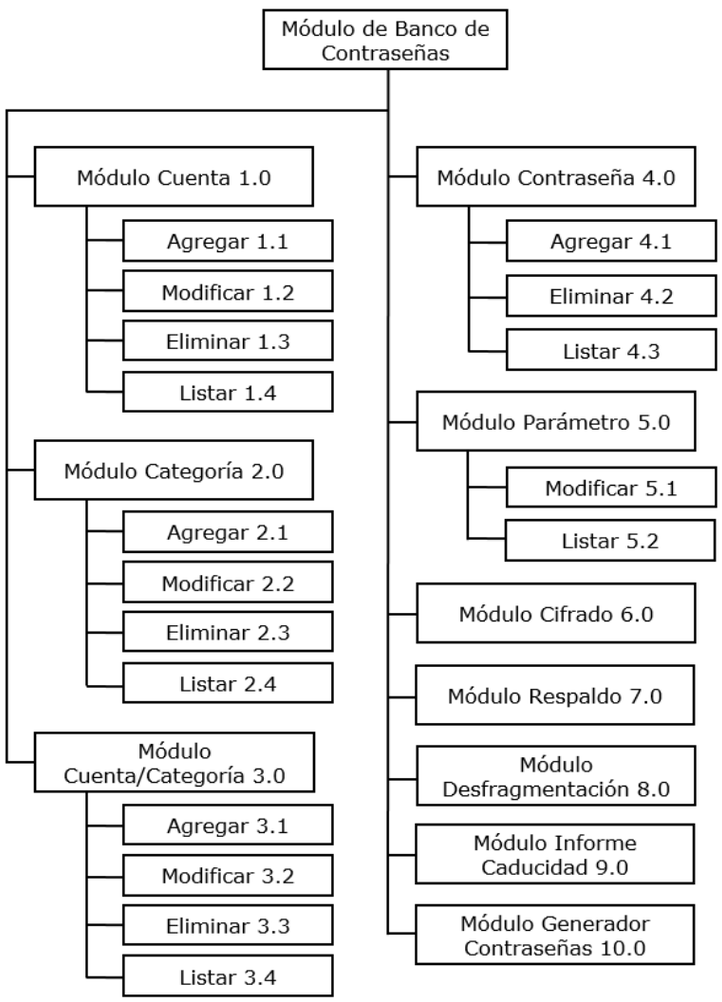

Figura 3-1. Diagrama modular del sistema.

Es importante destacar que no todos los módulos, presentados en el diagrama anterior, corresponden a una pantalla diferente. Los módulos de Cifrado y Desfragmentación no poseen ninguna representación visual, los módulos de Informe Caducidad, Generador de Contraseñas, Cuenta/Categoría y Contraseña se representan por ciertos elementos dentro de otras pantallas, tales como botones, imágenes, entradas de texto, entre otros, pero no necesariamente por pantallas propias. Es recomendable utilizar el diagrama de menús, de la siguiente sección, para observar las distintas pantallas que integra el sistema.

3.2. DIAGRAMA DE MENÚS

Existen tres tipos de menús en la aplicación, el primero, conocido como menú de la aplicación, es accesible por todas las diversas pantallas, exceptuando la de inicio de sesión y la de creación de la llave maestra, y tiene la funcionalidad de navegar al usuario por las principales ventanas del sistema, el segundo, conocido como submenú de la aplicación, se define de acuerdo a la pantalla que el usuario tiene activada en el momento, permitiéndole moverse a ventanas más específicas o realizar funciones acordes a dicha pantalla, y el tercero, conocido como menú contextual, corresponde a un menú que hace aparición al momento de hacer un click prolongado sobre un elemento de la pantalla, otorgando ciertas funcionalidades enlazadas con dicho elemento.

A continuación, con el objetivo de clarificar la navegabilidad del sistema, se muestra el diseño del menú de aplicación, el cual es accesible al hacer click sobre la imagen de las tres líneas blancas horizontales, en la esquina superior derecha:

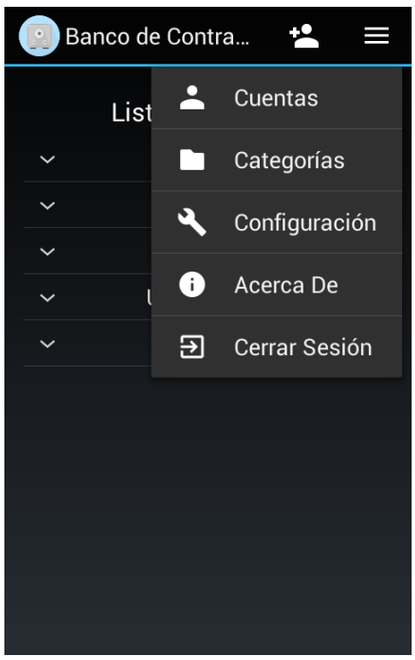

Figura 3-2. Diseño del Menú de Aplicación.

Además, se muestra un ejemplo del diseño del submenú de aplicación, específicamente el submenú para la pantalla de Consultar Cuenta, el cual es accesible al hacer click sobre la imagen del triángulo blanco, en la esquina superior derecha:

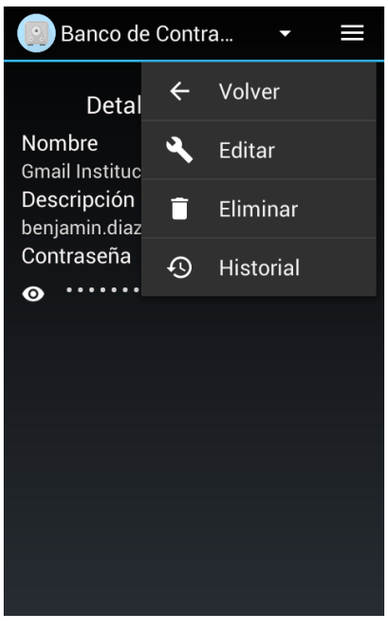

Figura 3-3. Diseño del Submenú de Aplicación, pantalla de Consultar Cuenta.

Cabe ser destacado que, en el caso de que solo haya un elemento dentro del submenú de aplicación, se reemplazará la imagen y funcionalidad del triángulo blanco, la cual abre el submenú, por la imagen y funcionalidad del elemento único.

El último diseño por mostrar corresponde a un ejemplo del menú contextual, específicamente el menú contextual que aparece al hacer un click prolongado sobre el nombre de una cuenta, en la pantalla Cuentas:

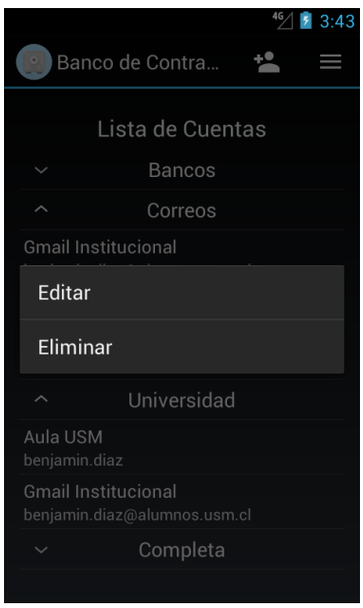

Figura 3-4. Diseño del Menú Contextual, pantalla Cuentas.

Ya para terminar, se muestra el diagrama de menús que presenta el sistema, destacando que dicho diagrama integra los tres tipos de menús mencionados anteriormente, y es único entre todos los diferentes usuarios:

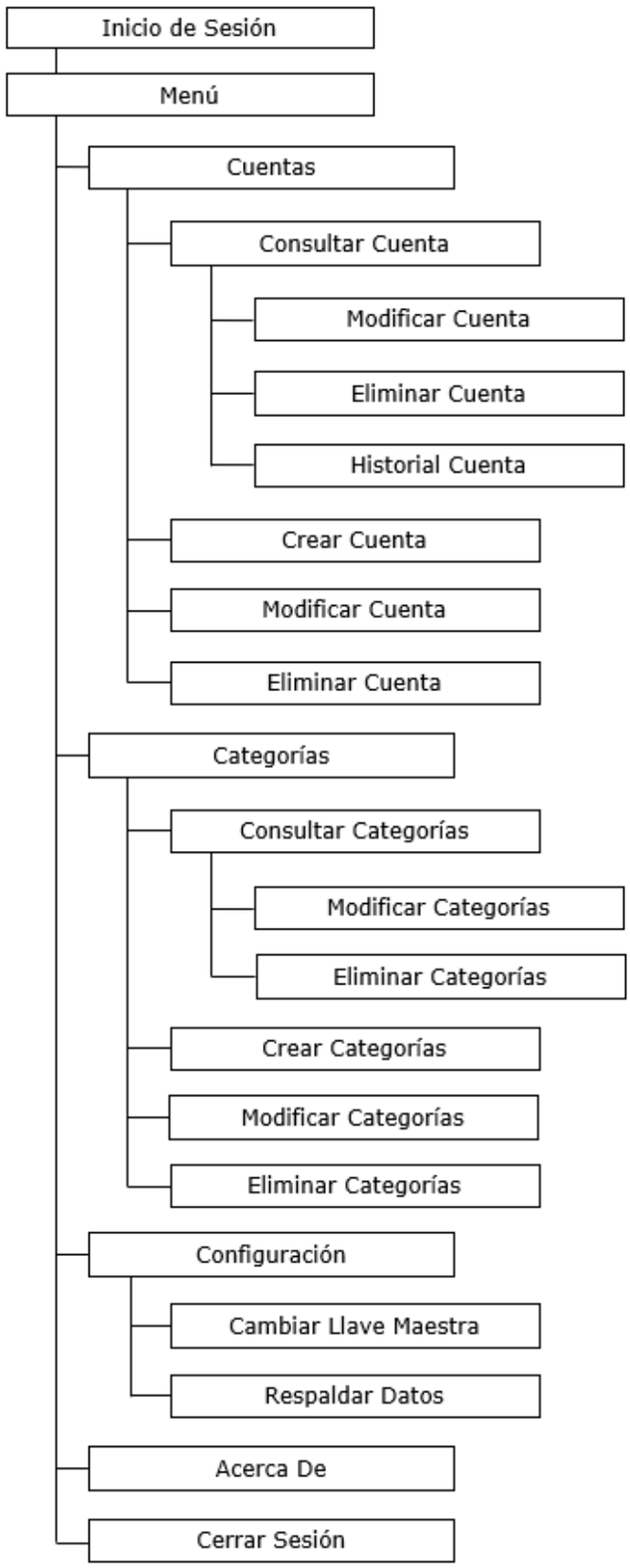

Figura 3-5. Diagrama de menús del sistema.

3.3. PROGRAMAS

A continuación, se presenta una lista de todos los diversos programas, o clases, que componen el sistema en cuestión. Además, se marca con un asterisco, *, las clases que serán descritas con más detalles, y cuyos códigos fuentes se encontrarán anexos al final del documento.

3.3.1. LISTA DE PROGRAMAS

	NOMBRE	OBJETIVO
1.	ActividadPrincipal.java*	Actividad principal de la aplicación, quien se encarga de cargar el Menú de Aplicación, se utiliza como contenedor de los diversos fragmentos existentes, los que se encargan de cargar el Submenú de Aplicación y el Menú Contextual.
2.	FragAcercaDe.java	Fragmento que muestra información general de la aplicación, como versión, año de creación, autor, etc.
3.	FragAgregarEditarCategoria.java	Fragmento que permite agregar una categoría, o editar los valores de una ya existente.
4.	FragAgregarEditarCuenta.java*	Fragmento que permite agregar una cuenta, o editar los valores de una ya existente.
5.	FragCambioLlaveMaestra.java*	Fragmento que le permite al usuario modificar su llave maestra.
6.	FragCategorias.java	Fragmento que contiene una lista de todas las categorías existentes, ordenadas por su campo POSICION, además permite modificar dicho campo.
7.	FragConfiguracion.java*	Fragmento que contiene una lista de los parámetros visibles de configuración de la aplicación, con la opción de que el usuario los pueda modificar.
8.	FragCrearLlaveMaestra.java	Fragmento que se muestra solo cuando la aplicación se instaló por primera vez, permite que el usuario cree su llave maestra, e importe un respaldo.

9.	FragCuentas.java*	Fragmento que contiene una lista de todas las categorías, ordenadas por su campo POSICION, en la que cada categoría puede desplegar una lista de todas sus cuentas asociadas, ordenadas por el campo POSICION, de la tabla CATEGORIA_CUENTA.
10.	FragDetalleCategoria.java*	Fragmento que contiene los datos asociados a una categoría, como el nombre, y una lista de sus cuentas asociadas, ordenadas por el campo POSICION, de la tabla CATEGORIA_CUENTA.
11.	FragDetalleCuenta.java	Fragmento que contiene los datos asociadas a una cuenta, como el nombre, descripción y última contraseña asociada.
12.	FragExportar.java*	Fragmento que permite respaldar la base de datos en la cuenta de Google Drive seleccionada por el usuario.
13.	FragHistorialCuenta.java	Fragmento que contiene una lista de todas las contraseñas asociadas a una cuenta determinada, ordenadas de más nueva a más antigua.
14.	FragInicioSesion.java*	Fragmento que solicita la llave maestra al usuario, para su posterior ingreso a los demás fragmentos.
15.	Cifrador.java	Encargarse de encriptar y desencriptar datos, obtener resultados hash y generar salts.
16.	Desfragmentador.java	Permitir la reutilización de IDs de contraseñas que ya no existen.
17.	GeneradorContrasennas.java	Encargarse de generar contraseñas, ya sea mediante caracteres aleatorios o palabras aleatorias, obtenidas de la base de datos “diccionario”, tabla PALABRA.
18.	CategoriaCuentaDAO.java	Permitir el acceso y manipulación de los datos de la tabla Categoría/Cuenta.
19.	CategoriaDAO.java	Permitir el acceso y manipulación de los datos de la tabla Categoría.
20.	ContrasennaDAO.java	Permitir el acceso y manipulación de los datos de la tabla Contraseña.
21.	CuentaDAO.java	Permitir el acceso y manipulación de los datos de la tabla Cuenta.

22.	PalabraDAO.java	Permitir el acceso a los datos de la tabla PALABRA, de la base de datos “diccionario”.
23.	ParametroDAO.java	Permitir el acceso y manipulación de los datos de la tabla Parámetro.
24.	DBOpenHelper.java	Administrar la creación, eliminación, configuración, actualización de la base de datos de la aplicación y su respaldo.
25.	DBOpenHelperDiccionario.java	Administrar la creación y actualización de la base de datos “diccionario”.
26.	ColCategoria.java	Entregar, al resto de programas, los nombres de las columnas de la tabla Categoría.
27.	ColCategoriaCuenta.java	Entregar, al resto de programas, los nombres de las columnas de la tabla Categoría/Cuenta.
28.	ColContrasenna.java	Entregar, al resto de programas, los nombres de las columnas de la tabla Contraseña.
29.	ColCuenta.java	Entregar, al resto de programas, los nombres de las columnas de la tabla Cuenta.
30.	ColParametro.java	Entregar, al resto de programas, los nombres de las columnas de la tabla Parámetro.
31.	Tabla.java	Entregar, al resto de programas, los nombres de las tablas de la base de datos de la aplicación.
32.	NombreBD.java	Entregar, al resto de programas, el nombre de los archivos que almacenan la base de datos de la aplicación y su respaldo.
33.	NombreParametro.java	Entregar, al resto de programas, el nombre de los parámetros de configuración, almacenados en la tabla Parámetro.
34.	Categoria.java	Representar una entidad Categoría.
35.	CategoriaCuenta.java	Representar una entidad Categoría/Cuenta.
36.	CategoriaListaCuentas.java	Representar una entidad Categoría, con su lista de objetos CuentaConFecha asociados.
37.	CategoriaSeleccionable.java	Representar una entidad Categoría, con la capacidad de ser seleccionada.
38.	Contrasenna.java	Representar una entidad Contraseña.
39.	Cuenta.java	Representar una entidad Cuenta.

40.	CuentaConFecha.java	Representar una entidad Cuenta, incluyendo la fecha de creación de su última contraseña.
41.	Parametro.java	Representar una entidad Parámetro.
42.	ParametroSeleccionable.java	Representar una entidad Parámetro, con la capacidad de ser seleccionada.
43.	AdapCategorias.java	Preparar una lista de objetos Categorías para su visualización.
44.	AdapCategoriasCheckBox.java	Preparar una lista de objetos CategoriaSeleccionable, con un checkbox para permitir su selección, para su visualización.
45.	AdapCategoriasCuentas.java	Preparar una lista de objetos CategoriaListaCuentas para su visualización.
46.	AdapContrasennas.java	Preparar una lista de objetos Contrasenna para su visualización.
47.	AdapCuentas.java	Preparar una lista de objetos Cuenta para su visualización.
48.	AdapParametros.java	Preparar una lista de objetos ParametroSeleccionable para su visualización.
49.	CustomToast.java	Mostrar un mensaje en pantalla, que desaparece automáticamente.
50.	ExcepcionBancoContrasennas.java	Controlar los errores ocurridos.

3.3.2. DESCRIPCIÓN DE PROGRAMAS

3.3.2.1. ActividadPrincipal.java Página en Anexos: 79

Nombre Lógico:

Actividad Principal.

Objetivo:

Actividad principal de la aplicación, quien se encarga de cargar el Menú de Aplicación, utilizada como contenedor de los diversos fragmentos existentes, los que se encargan de cargar el Submenú de Aplicación y el Menú Contextual.

Diagrama de Bloques:

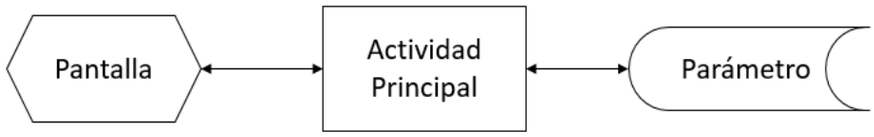

Figura 3-6. Diagrama de bloques - Actividad Principal.

Reglas del Proceso:

1. Se presenta parte de la pantalla de la figura 3-7, es decir, la barra superior, con el nombre de la aplicación, el submenú y menú de aplicación, además del contenedor de fragmentos.
2. Carga en pantalla el fragmento correspondiente al Inicio de Sesión, si no se ha iniciado sesión, o el último fragmento utilizado, si la sesión está abierta.
3. En caso de minimizar la aplicación, controla el cierre de sesión automático, viendo la diferencia de fecha/hora entre que se minimizó y se maximizó el programa.
4. En caso de interactuar con el menú de aplicación, controla el fragmento que se carga en el contenedor.
5. En caso de apretar Volver en el submenú de aplicación, o en el botón del dispositivo móvil, controla el fragmento que se debe cargar en el contenedor.

Diseño de Pantalla:

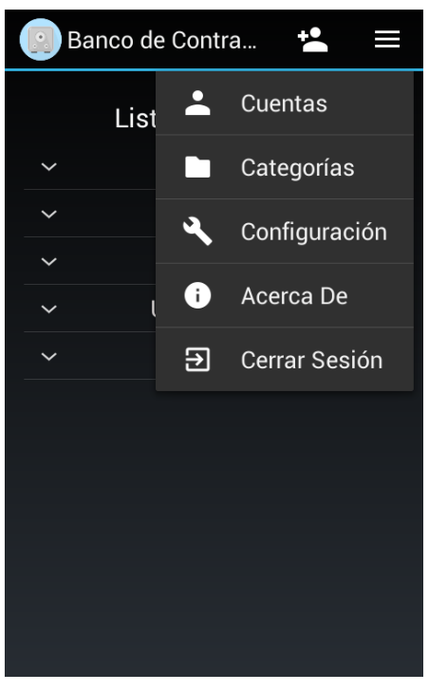

Figura 3-7. Diseño de la Actividad Principal, con el Menú de Aplicación.

3.3.2.2. FragAgregarEditarCuenta.java Página en Anexos: 90

Nombre Lógico:

Agregar/Editar Cuenta.

Objetivo:

Fragmento que permite agregar una cuenta, o editar los valores de una ya existente.

Diagrama de Bloques:

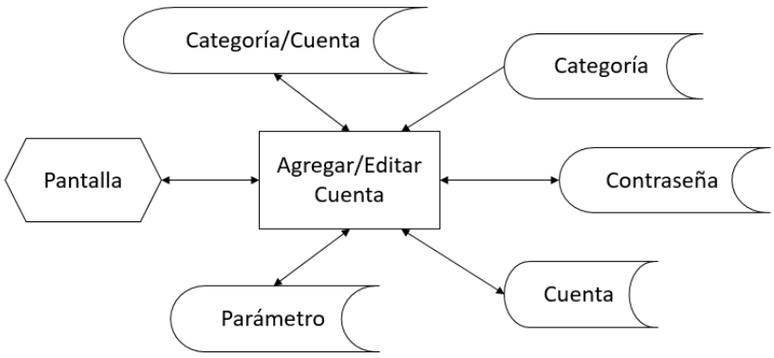

Figura 3-8. Diagrama de bloques – Agregar/Editar Cuenta.

Reglas del Proceso:

1. Se presenta la pantalla de la figura 3-9.
2. Se despliega la información de la cuenta y su última contraseña, si se está editando una cuenta, y un listado de las categorías existentes.
3. Se selecciona los checkbox para las categorías asociadas a la cuenta editada.
4. Se define el modo del generador de contraseñas, de acuerdo a la tabla Parámetro, ya sea por caracteres o palabaras aleatorias.
5. El usuario ingresa el nombre, descripción, contraseña, tiempo de validez de la contraseña y selecciona las categorías asociadas a la cuenta, y si usa el generador para ingresar la contraseña, se graba en la tabla parámetros cual fue la última modalidad usada.
6. El usuario selecciona Guardar en el Submenú de Aplicación.
7. Se valida que el nombre y la contraseña no estén vacíos, que la validez, que representa la cantidad de días antes de caducar una contraseña, sea un valor numérico mayor o igual a 0 y que no exista otra cuenta con el mismo nombre.
8. Se inserta o actualiza la cuenta deseada.
9. Se obtiene la fecha actual, y si la contraseña ingresada no está registrada, se encripta usando el algoritmo AES, en modo CBC, y se inserta junto a la fecha obtenida.

10. Por cada categoría, se revisa si está relacionada con la cuenta, y dependiendo de ello, y si el usuario la seleccionó, se crea o elimina la relación Categoría/Cuenta.
11. Se informa al usuario que la cuenta fue creada o modificada exitosamente, y se carga la pantalla de Lista de Cuentas.

Diseño de Pantalla:

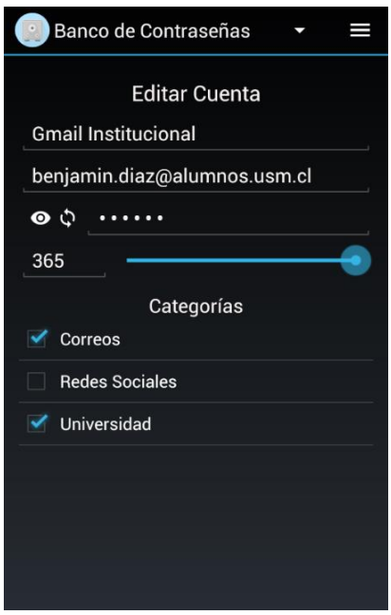

Figura 3-9. Diseño de la pantalla Agregar/Editar Cuenta.

3.3.2.3. FragCambioLlaveMaestra.java Página en Anexos: 104

Nombre Lógico:

Cambiar Llave Maestra.

Objetivo:

Fragmento que le permite al usuario modificar su llave maestra.

Diagrama de Bloques:

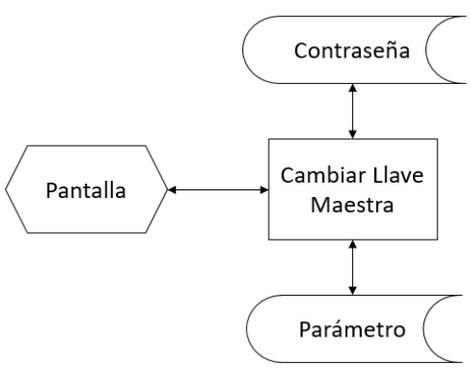

Figura 3-10. Diagrama de bloques – Cambiar Llave Maestra.

Reglas del Proceso:

1. Se presenta la pantalla de la figura 3-11.
2. El usuario ingresa su llave maestra actual, su nueva llave maestra y la confirma con un segundo ingreso.
3. El usuario selecciona Guardar en el Submenú de Aplicación.
4. Se valida que la llave maestra actual no esté vacía, que coincida con la llave maestra almacenada por la aplicación, en la tabla Parámetro, se valida que la nueva llave maestra no esté vacía, que su largo no sea menor al mínimo permitido, que su confirmación coincida con ésta, y que no sea igual a la llave maestra actual.
5. Se obtiene una clave de encriptación en base a la nueva llave maestra.
6. Por cada contraseña, se reencripta usando la nueva clave de encriptación.
7. Se actualizan los valores asociados a la llave maestra, en la tabla Parámetro, con los valores asociados a la nueva llave maestra seleccionada por el usuario.
8. Se informa al usuario que su llave maestra fue actualizada exitosamente, y se carga la pantalla de Configuración.

Diseño de Pantalla:

Figura 3-11. Diseño de la pantalla Cambiar Llave Maestra.

3.3.2.4. FragConfiguracion.java Página en Anexos: 109

Nombre Lógico:

Configuración.

Objetivo:

Fragmento que contiene una lista de los parámetros visibles de configuración de la aplicación, con la opción de que el usuario los pueda modificar.

Diagrama de Bloques:

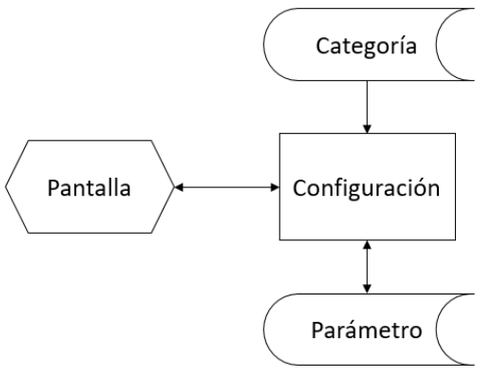

Figura 3-12. Diagrama de bloques – Configuración.

Reglas del Proceso:

1. Se presenta la pantalla de la figura 3-13.
2. Se despliega una lista de los parámetros visibles.
3. El usuario ingresa los nuevos valores de los parámetros que desea modificar.
4. El usuario selecciona Guardar en el Submenú de Aplicación.
5. Se validan los parámetros ingresados, que no estén vacíos, que sean numéricos si corresponde y que su valor no coincida con alguna restricción de la base de datos, como por ejemplo, el nombre de la categoría global no puede encontrarse en la tabla Categoría.
6. Se actualiza cada uno de los parámetros en la base de datos.
7. Se muestran los nuevos valores almacenados.
8. Se informa al usuario que los cambios efectuados, sobre los parámetros, fueron grabados exitosamente.

Diseño de Pantalla:

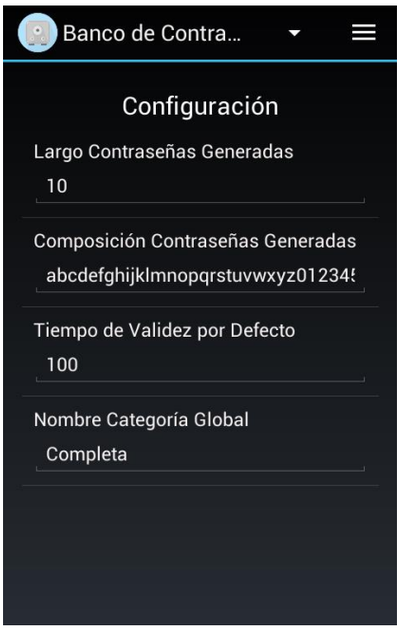

Figura 3-13. Diseño de la pantalla Configuración.

3.3.2.5. FragCuentas.java Página en Anexos: 113

Nombre Lógico:

Lista de Cuentas.

Objetivo:

Fragmento que contiene una lista de todas las categorías, ordenadas por su campo POSICION, en la que cada categoría puede desplegar una lista de todas sus cuentas asociadas, ordenadas por el campo POSICION, de la tabla CATEGORIA_CUENTA. Se puede seleccionar una cuenta para ver su detalle, para modificarla o eliminarla.

Diagrama de Bloques:

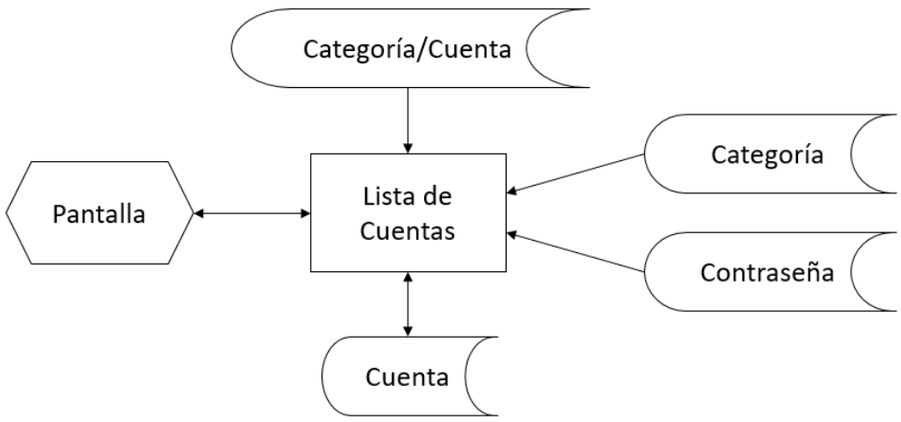

Figura 3-14. Diagrama de bloques – Lista de Cuentas.

Reglas del Proceso:

1. Se presenta la pantalla de la figura 3-15.
2. Se despliega una lista de las categorías, cada una con su propia lista de cuentas asociadas, y se calcula la necesidad de actualizar la contraseña de cada cuenta, para mostrar el icono que corresponde junto al nombre de la cuenta.
3. El usuario selecciona una categoría de interés.
4. Se presenta la lista de cuentas asociadas a la categoría seleccionada.
5. El usuario selecciona la cuenta de interés, además puede elegir modificar una cuenta o eliminarla.
6. Se prepara la pantalla de Detalle de Cuenta, para la cuenta seleccionada.
7. Se carga la pantalla de Detalle de Cuenta.

Diseño de Pantalla:

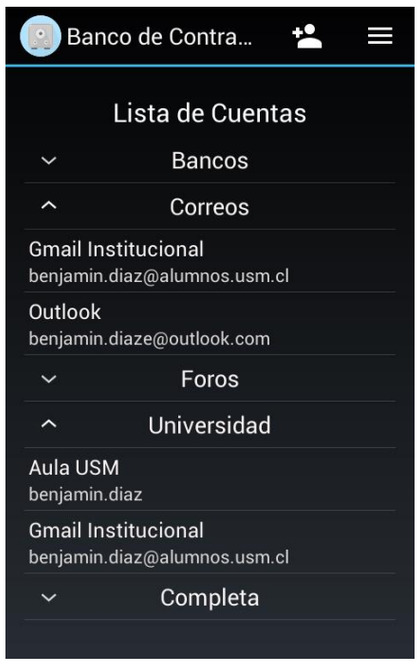

Figura 3-15. Diseño de la pantalla Lista de Cuentas.

3.3.2.6. FragDetalleCategoria.java Página en Anexos: 118

Nombre Lógico:

Detalle de Categoría.

Objetivo:

Fragmento que contiene los datos asociados a una categoría, como el nombre, y una lista de sus cuentas asociadas, ordenadas por el campo POSICION, de la tabla CATEGORIA_CUENTA.

Diagrama de Bloques:

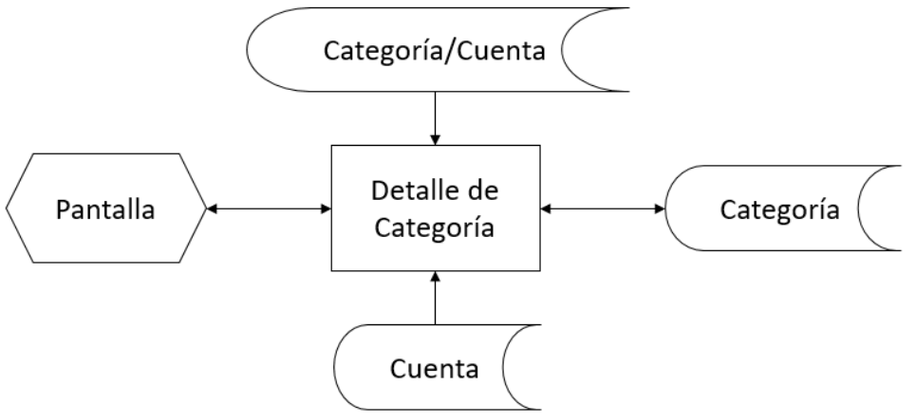

Figura 3-16. Diagrama de bloques – Detalle de Categoría.

Reglas del Proceso:

1. Se presenta la pantalla de la figura 3-17.
2. Se despliega la información de la categoría seleccionada, y un listado de las cuentas asociadas a dicha categoría.
3. Se permite editar o eliminar la categoría, con el Submenú de Aplicación, y cambiar la posición de las cuentas asociadas a la categoría. Al eliminar una categoría, se emite un mensaje explicando que las cuentas asociadas no se eliminarán, y podrán ser encontradas en la categoría por omisión.

Diseño de Pantalla:

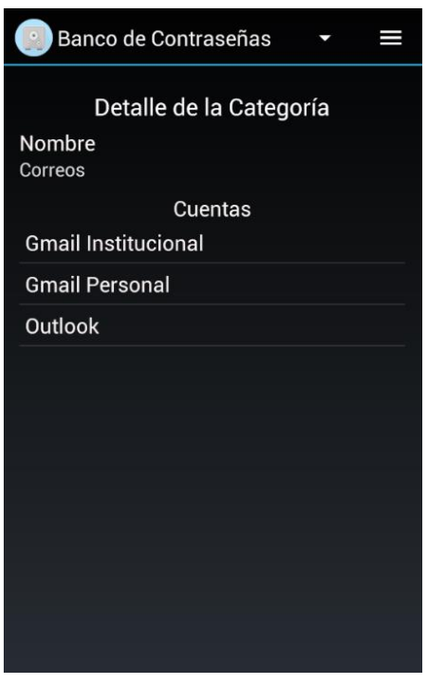

Figura 3-17. Diseño de la pantalla Detalle de Categoría.

3.3.2.7. FragExportar.java Página en Anexos: 122

Nombre Lógico:

Respaldar Datos.

Objetivo:

Fragmento que permite respaldar la base de datos en la cuenta de Google Drive seleccionada por el usuario.

Diagrama de Bloques:

Figura 3-18. Diagrama de bloques – Respaldar Datos.

Reglas del Proceso:

1. Se presenta la pantalla de la figura 3-19.
2. Se despliega la cuenta Google del usuario, si está almacenada en la tabla Parámetro.
3. El usuario selecciona la cuenta Google en la que quiere respaldar sus datos.
4. El usuario selecciona Respaldar en el Submenú de Aplicación.
5. Se valida que la cuenta seleccionada no esté vacía.
6. Se deshabilita la rotación de pantalla.
7. Se actualiza la cuenta Google en la base de datos, tabla Parámetro.
8. Se conecta la aplicación con la API de Google Drive.
9. Se obtiene la lista de respaldos almacenados en el directorio de aplicación de Google Drive.
10. Se crea un nuevo contenido en el directorio de aplicación de Google Drive.
11. Se llena dicho contenido con los datos del archivo local correspondiente a la base de datos, y se le configura sus metadatos.
12. Se crea un archivo nuevo en el directorio de aplicación de Google Drive, con el nuevo contenido generado.
13. Se eliminan los respaldos anteriores del directorio de aplicación de Google Drive.
14. Se informa al usuario que su respaldo fue creado exitosamente.

Diseño de Pantalla:

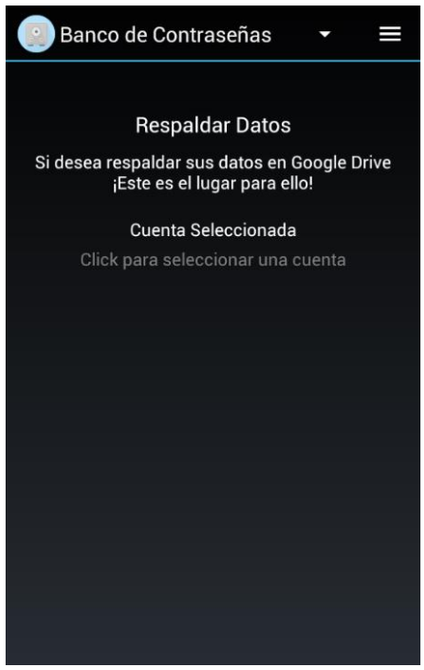

Figura 3-19. Diseño de la pantalla Respaldar Datos.

3.3.2.8. FragInicioSesion.java Página en Anexos: 132

Nombre Lógico:

Inicio de Sesión.

Objetivo:

Fragmento que solicita la llave maestra al usuario, para su posterior ingreso a los demás fragmentos.

Diagrama de Bloques:

Figura 3-20. Diagrama de bloques – Inicio de Sesión.

Reglas del Proceso:

1. Se presenta la pantalla de la figura 3-21.
2. Se crea la base de datos del diccionario de palabras si no existe.
3. Se desfragmenta la base de datos si se han creado mil contraseñas nuevas desde la última desfragmentación.

4. El usuario ingresa su llave maestra, y selecciona Ingresar.
5. Se valida que la llave maestra no esté vacía.
6. Se obtiene la sal y resultado hash de la base de datos, tabla Parámetro.
7. Se obtiene el resultado hash de la llave maestra ingresada con la sal obtenida de la base de datos.
8. Se valida que el resultado hash almacenado en la tabla Parámetro sea igual al resultado hash obtenido con la llave maestra ingresada.
9. Se obtiene la sal de encriptación de la base de datos, y se genera la clave de encriptación utilizada para trabajar con las contraseñas.
10. Se inicia sesión, lo cual carga la pantalla de Lista de Cuentas.

Diseño de Pantalla:

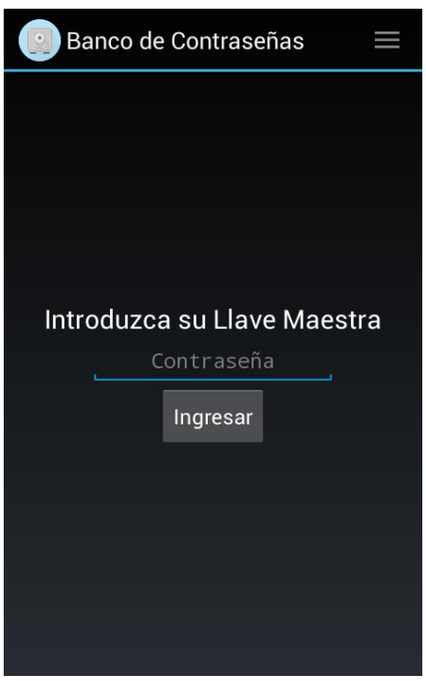

Figura 3-21. Diseño de la pantalla Inicio de Sesión.

CONCLUSIONES Y RECOMENDACIONES

El objetivo de todo sistema informático consiste en mejorar la eficiencia de un proceso que se lleva a cabo, ya sea, por ejemplo, mejorando la rapidez del proceso, eliminando fuentes de errores, o limitando accesos a personas no autorizadas. Por lo que, mientras existan procesos que se puedan mejorar, existirán posibles sistemas informáticos que den soluciones a dichas falencias.

Para poder construir un sistema informático íntegro, es necesario llevar a cabo cada una de las etapas que pertenecen al ciclo de vida de éste, desde el análisis y diseño, hasta la implementación y mantenimiento.

Durante el último año de la carrera, uno se ve en la situación de pasar por parte de dicho ciclo de vida para el desarrollo de su trabajo de título, desde el análisis y diseño, hasta la construcción y pruebas, pero ninguna de estas etapas es nueva para uno, pues durante el transcurso de la carrera, uno va aprendiendo y poniendo en práctica cada una de las etapas mencionadas, gracias a los diversos lenguajes de programación enseñados y las diversas actividades que uno debe ir completando. Por lo que es altamente recomendable el utilizar el trabajo de título como una nueva actividad para aprender cosas nuevas, mientras se practica lo ya aprendido.

Dicho lo anterior, y sin menospreciar la utilización de tecnologías ya vistas durante el transcurso del periodo estudiantil, es una muy buena idea el utilizar esta oportunidad para aprender de tecnologías nuevas, talvez incluso si esto signifique el desarrollo de un sistema informático más pequeño de lo normal, con tal de enfocarse primeramente en el aprendizaje que dichas tecnologías conllevan. Por lo cual un sistema informático pequeño, como el Banco de Contraseñas, es perfecto para cumplir dicho objetivo, en paralelo a que resuelve una serie de problemáticas que afectan a la sociedad actual.

Gracias al desarrollo del sistema Banco de Contraseñas, se aprendió a trabajar con otras tecnologías, como la IDE de Android Studio, a integrar dichas tecnologías con desarrollos construidos por terceros, como la API de Google Drive, y a poner en práctica una serie de metodologías criptográficas que mantengan segura la información sensible de los clientes, todos aspectos que no se abordaron profundamente durante los años de estudio en la universidad, debido, en gran parte, a que se dio prioridad a otros temas igual de importantes, pero gracias a esto, uno puede llegar a obtener su título universitario, cumpliendo con los requisitos solicitados por la universidad, en conjunto a los requisitos extracurriculares que un proyecto como éste traen consigo.

BIBLIOGRAFÍA

- Trabajo de Título “Gestión de Venta y Control de Stock Ferretería Zamora”, por Esteban Godoy y Bastián Zamora, año 2014, UTFSM, Sede Viña del Mar.

- “Best Password Manager For Android”, por Andrew Martonik – Android Central, 25 Septiembre 2017.
  https://www.androidcentral.com/best-password-managers-android

- Paneles de Control, Versiones de la plataforma, por Android Developers, 05/06/2017.
  https://developer.android.com/about/dashboards/index.html?hl=es-419

- “Worldwide Smartphone Sales to End Users by Operating System in 4Q16”, por Gartner, Febrero 2017.
  https://www.gartner.com/newsroom/id/3609817

- Guías de Android, por Android Developers.
  https://developer.android.com/guide/index.html?hl=es-419

- Guías de Google Drive Android API, por Google, 12 Marzo 2018.
  https://developers.google.com/drive/android/intro

- “Google Drive en Android” por sgoliver, 27/09/2016.
  http://www.sgoliver.net/blog/google-drive-en-android-1/

- “Generate Secure Password Hash: MD5, SHA, PBKDF2, BCrypt Examples”, por Lokesh Gupta, 22 Julio 2013, HowToDoInJava.
  https://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/

ANEXOS

ActividadPrincipal.java 1/11

```
1   package cl.theroot.passbank;
2
3   import android.app.Activity;
4   import android.app.AlertDialog;
5   import android.app.Fragment;
6   import android.app.FragmentManager;
7   import android.app.FragmentTransaction;
8   import android.content.Context;
9   import android.content.DialogInterface;
10  import android.content.Intent;
11  import android.os.Bundle;
12  import android.util.Log;
13  import android.view.Menu;
14  import android.view.MenuInflater;
15  import android.view.MenuItem;
16  import android.view.View;
17  import android.view.inputmethod.InputMethodManager;
18
19  import java.util.ArrayList;
20  import java.util.Calendar;
21  import java.util.List;
22
23  import cl.theroot.passbank.dominio.Categoria;
24  import cl.theroot.passbank.dominio.Cuenta;
25  import cl.theroot.passbank.dominio.Parametro;
26  import cl.theroot.passbank.fragmento.FragAcercaDe;
27  import cl.theroot.passbank.fragmento.FragAgregarEditarCategoria;
28  import cl.theroot.passbank.fragmento.FragAgregarEditarCuenta;
29  import cl.theroot.passbank.fragmento.FragCambioLlaveMaestra;
30  import cl.theroot.passbank.fragmento.FragCategorias;
31  import cl.theroot.passbank.fragmento.FragConfiguracion;
32  import cl.theroot.passbank.fragmento.FragCrearLlaveMaestra;
33  import cl.theroot.passbank.fragmento.FragCuentas;
34  import cl.theroot.passbank.fragmento.FragDetalleCategoria;
35  import cl.theroot.passbank.fragmento.FragDetalleCuenta;
36  import cl.theroot.passbank.fragmento.FragExportar;
37  import cl.theroot.passbank.fragmento.FragHistorialCuenta;
38  import cl.theroot.passbank.fragmento.FragInicioSesion;
39  import cl.theroot.passbank.datos.ParametroDAO;
40  import cl.theroot.passbank.datos.nombres.ColCategoria;
41  import cl.theroot.passbank.datos.nombres.ColCuenta;
42  import cl.theroot.passbank.datos.nombres.NombreParametro;
43
44  public class ActividadPrincipal extends Activity {
45      private static final String TAG = "BdC-ActividadPrincipal";
46      private FragmentManager administradorFragmentos =
        getFragmentManager();
47      private Fragment fragmentoActual;
48
49      private final String estadoLogin = "Sesión Iniciada";
50      private final String onStopTime = "Tiempo Stop";
51      private final String historialFragmentos = "Historial Fragmentos";
52      private final String historialBundles = "Historial Argumentos";
53      private final String ultimoFragmento = "Último Fragmento";
54      private final String ultimoBundle = "Último Bundle";
```

ActividadPrincipal.java 2/11

```
55
56      private boolean sesionIniciada;
57      private Long onStop;
58
59      private final int waitTime = 30 * 1000;
60
61      private final List<Class> listaFragmentos = new ArrayList();
62      private final List<Bundle> listaBundles = new ArrayList();
63
64      public static String llaveEncrip = null;
65      private Menu menu;
66
67      @Override
68      protected void onCreate(Bundle savedInstanceState) {
69          super.onCreate(savedInstanceState);
70          Log.i(TAG, "onCreate():");
71
72          setContentView(R.layout.actividad_principal);
73
74          if (savedInstanceState != null) {
75              sesionIniciada = savedInstanceState.getBoolean(
                estadoLogin);
76              onStop = savedInstanceState.getLong(onStopTime);
77              stringsToListaFragmentos(savedInstanceState.
                getStringArrayList(historialFragmentos));
78              stringsToListaBundles(savedInstanceState.
                getStringArrayList(historialBundles));
79              Class clase = stringToClass(savedInstanceState.getString(
                ultimoFragmento));
80              Bundle bundle = stringToBundle(savedInstanceState.
                getString(ultimoBundle));
81              try {
82                  Fragment fragmento = (Fragment) clase.getConstructor
                    ().newInstance();
83                  fragmento.setArguments(bundle);
84                  fragmentoActual = fragmento;
85              } catch (Exception ex) {
86                  Log.e(TAG, "Error al tratar de recrear el fragmento
                    actual", ex);
87              }
88          } else {
89              userLogOut();
90          }
91          Log.i(TAG, "---" + estadoLogin + ": " + sesionIniciada);
92          Log.i(TAG, "---" + onStopTime + ": " + onStop);
93      }
94
95      @Override
96      public void onSaveInstanceState(Bundle savedInstanceState) {
97          if (onStop == null) {
98              onStop = Calendar.getInstance().getTimeInMillis();
99          }
100         Log.i(TAG, "onSaveInstanceState: ");
101         Log.i(TAG, "---" + estadoLogin + ": " + sesionIniciada);
102         Log.i(TAG, "---" + onStopTime + ": " + onStop);
```

```
103         savedInstanceState.putBoolean(estadoLogin, sesionIniciada);
104         savedInstanceState.putLong(onStopTime, onStop);
105
106         //Almacenar las listas de historiales
107         savedInstanceState.putStringArrayList(historialFragmentos,
            listaFragmentoToStrings());
108         savedInstanceState.putStringArrayList(historialBundles,
            listaBundlesToStrings());
109
110         //Almacenar el fragmento actual
111         savedInstanceState.putString(ultimoFragmento, classToString(
            fragmentoActual.getClass()));
112         savedInstanceState.putString(ultimoBundle, bundleToString(
            fragmentoActual.getArguments()));
113
114         super.onSaveInstanceState(savedInstanceState);
115     }
116
117     @Override
118     public void onActivityResult(final int requestCode, final int
        resultCode, final Intent data) {
119         super.onActivityResult(requestCode, resultCode, data);
120         Fragment frg = getFragmentManager().findFragmentById(R.id.
            FL_contenedor);
121         if (frg != null) {
122             frg.onActivityResult(requestCode, resultCode, data);
123         }
124     }
125
126     @Override
127     public void onStop() {
128         Log.i(TAG, "onStop():");
129         onStop = Calendar.getInstance().getTimeInMillis();
130         Log.i(TAG, "---onStop: " + onStop);
131         super.onStop();
132     }
133
134     @Override
135     public void onStart() {
136         Log.i(TAG, "onStart():");
137         if (onStop != null) {
138             long onStart = Calendar.getInstance().getTimeInMillis();
139             long diff = Math.abs(onStart - onStop);
140             Log.i(TAG, "---onStart: " + onStart + " onStop: " +
                onStop + " Diferencia: " + diff);
141             onStop = null;
142             if (diff > waitTime) {
143                 userLogOut();
144             }
145         }
146         super.onStart();
147     }
148
149     @Override
150     public void onBackPressed() {
```

ActividadPrincipal.java 4/11

```
151         if (listaFragmentos.isEmpty()) {
152             AlertDialog alertDialog = new AlertDialog.Builder(this).
                create();
153             alertDialog.setTitle("Cerrar la Aplicación");
154             alertDialog.setMessage("Está a punto de cerrar la
                aplicación, ¿Desea continuar?");
155             alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "NO",
                new DialogInterface.OnClickListener() {
156                 public void onClick(DialogInterface dialog, int which
                    ) {
157                     dialog.dismiss();
158                 }
159             });
160             alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "SÍ",
                new DialogInterface.OnClickListener() {
161                 public void onClick(DialogInterface dialog, int which
                    ) {
162                     finish();
163                     dialog.dismiss();
164                 }
165             });
166             alertDialog.show();
167             return;
168         } else {
169             Class clase = listaFragmentos.get(listaFragmentos.size()
                - 1);
170             Bundle bundle = listaBundles.get(listaBundles.size() - 1);
171             if (clase == FragInicioSesion.class) {
172                 AlertDialog alertDialog = new AlertDialog.Builder(
                    this).create();
173                 alertDialog.setTitle("Cerrar Sesión");
174                 alertDialog.setMessage("Está a punto de cerrar
                    sesión, ¿Desea continuar?");
175                 alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE,
                    "NO", new DialogInterface.OnClickListener() {
176                     public void onClick(DialogInterface dialog, int
                        which) {
177                         dialog.dismiss();
178                     }
179                 });
180                 alertDialog.setButton(AlertDialog.BUTTON_POSITIVE,
                    "SÍ", new DialogInterface.OnClickListener() {
181                     public void onClick(DialogInterface dialog, int
                        which) {
182                         userLogOut();
183                         dialog.dismiss();
184                     }
185                 });
186                 alertDialog.show();
187                 return;
188             }
189             listaFragmentos.remove(listaFragmentos.size() - 1);
190             listaBundles.remove(listaBundles.size() - 1);
191             try {
```

```
192                 Fragment fragmento = (Fragment) clase.getConstructor
                    ().newInstance();
193                 fragmento.setArguments(bundle);
194
195                 View view = findViewById(R.id.FL_contenedor);
196                 InputMethodManager inputMethodManager = (
                    InputMethodManager) getSystemService(Context.
                    INPUT_METHOD_SERVICE);
197                 inputMethodManager.hideSoftInputFromWindow(view.
                    getWindowToken(), InputMethodManager.HIDE_NOT_ALWAYS);
198
199                 FragmentTransaction fragmentTransaction =
                    administradorFragmentos.beginTransaction();
200                 fragmentTransaction.replace(R.id.FL_contenedor,
                    fragmento);
201                 fragmentTransaction.commit();
202                 fragmentoActual = fragmento;
203                 System.out.println(listaFragmentos);
204             } catch(Exception ex) {
205                 ex.printStackTrace();
206             }
207         }
208         //super.onBackPressed();
209     }
210
211     //Creación del menú de la aplicación
212     @Override
213     public boolean onCreateOptionsMenu(Menu menu) {
214         this.menu = menu;
215         MenuInflater inflater = getMenuInflater();
216         inflater.inflate(R.menu.menu_actividad, menu);
217         return super.onCreateOptionsMenu(menu);
218     }
219
220     //Cuando se prepare el menú de la aplicación
221     @Override
222     public boolean onPrepareOptionsMenu(Menu menu) {
223         Log.d(TAG, "onPrepareOptionsMenu executed!");
224         if (!sesionIniciada) {
225             menu.close();
226             menu.findItem(R.id.menu).setVisible(false);
227         } else {
228             menu.findItem(R.id.menu).setVisible(true);
229         }
230         return super.onPrepareOptionsMenu(menu);
231     }
232
233     //Funcionalidad del menú principal de la aplicación
234     @Override
235     public boolean onOptionsItemSelected(MenuItem item) {
236         //Si la sesión no está iniciada, no se hace nada.
237         if (!sesionIniciada) {
238             return false;
239         }
240         switch (item.getItemId()) {
```

```
241             case R.id.menu_accounts:
242                 return cambiarFragmento(new FragCuentas());
243
244             case R.id.menu_categories:
245                 return cambiarFragmento(new FragCategorias());
246
247             case R.id.menu_configuration:
248                 return cambiarFragmento(new FragConfiguracion());
249
250             case R.id.menu_about:
251                 return cambiarFragmento(new FragAcercaDe());
252
253             case R.id.menu_logout:
254                 userLogOut();
255                 return true;
256
257             default:
258                 return super.onOptionsItemSelected(item);
259         }
260     }
261
262     public boolean cambiarFragmento(Fragment fragmento) {
263         View view = findViewById(R.id.FL_contenedor);
264         InputMethodManager inputMethodManager = (InputMethodManager)
            getSystemService(Context.INPUT_METHOD_SERVICE);
265         inputMethodManager.hideSoftInputFromWindow(view.
            getWindowToken(), InputMethodManager.HIDE_NOT_ALWAYS);
266
267         FragmentTransaction fragmentTransaction =
            administradorFragmentos.beginTransaction();
268         fragmentTransaction.replace(R.id.FL_contenedor, fragmento);
269
270         if (fragmentoActual != null && fragmentoActual instanceof
            Fragment) {
271             /*if (listaFragmentos.size() >= 10) {
272                 listaFragmentos.remove(0);
273                 listaBundles.remove(0);
274             }*/
275             if (listaFragmentos.size() == 0 || fragmentoActual.
                getClass() != listaFragmentos.get(listaFragmentos.size()
                - 1)) {
276                 listaFragmentos.add(fragmentoActual.getClass());
277                 listaBundles.add(fragmentoActual.getArguments());
278             }
279         }
280         if (fragmento instanceof FragInicioSesion) {
281             listaFragmentos.clear();
282             listaBundles.clear();
283         }
284
285         fragmentTransaction.commit();
286         fragmentoActual = fragmento;
287
288         Log.i(TAG, listaFragmentos.toString());
289         return true;
```

ActividadPrincipal.java 7/11

```
290     }
291
292     public void actualizarBundles(Class clase, String antiguoValor,
        String nuevoValor) {
293         List<Class> reqCatName = new ArrayList<>();
294         reqCatName.add(FragAgregarEditarCategoria.class);
295         reqCatName.add(FragDetalleCategoria.class);
296
297         List<Class> reqCueName = new ArrayList<>();
298         reqCueName.add(FragAgregarEditarCuenta.class);
299         reqCueName.add(FragDetalleCuenta.class);
300         reqCueName.add(FragHistorialCuenta.class);
301
302         if (nuevoValor != null) {
303             Log.i(TAG, "Actualizando bundles...");
304             for (int i = 0; i < listaFragmentos.size(); i++) {
305                 Log.i(TAG, "Checkeando fragmento: " + listaFragmentos
                    .get(i));
306                 if ((clase == Categoria.class && reqCatName.contains(
                    listaFragmentos.get(i))) || (clase == Cuenta.class &&
                     reqCueName.contains(listaFragmentos.get(i)))) {
307                     if (listaBundles.get(i) != null) {
308                         if ((clase == Categoria.class && listaBundles
                            .get(i).getString(ColCategoria.NOMBRE.
                            toString()).equals(antiguoValor)) || (clase
                            == Cuenta.class && listaBundles.get(i).
                            getString(ColCuenta.NOMBRE.toString()).equals
                            (antiguoValor))) {
309                             Log.i(TAG, "Cambiando bundle...");
310                             Bundle nuevo = new Bundle();
311                             if (clase == Categoria.class) {
312                                 nuevo.putString(ColCategoria.NOMBRE.
                                    toString(), nuevoValor);
313                             } else if (clase == Cuenta.class) {
314                                 nuevo.putString(ColCuenta.NOMBRE.
                                    toString(), nuevoValor);
315                             }
316                             listaBundles.set(i, nuevo);
317                         }
318                     }
319                 }
320             }
321         } else {
322             Log.i(TAG, "Eliminando bundles...");
323             List<Class> nuevaListaFragmentos = new ArrayList<>(
                listaFragmentos);
324             List<Bundle> nuevaListaBundles = new ArrayList<>(
                listaBundles);
325             Log.i(TAG, "Limpiando lista de fragmentos...");
326             listaFragmentos.clear();
327             listaBundles.clear();
328             for (int i = 0; i < nuevaListaFragmentos.size(); i++) {
329                 Log.i(TAG, "Checkeando fragmento: " +
                    nuevaListaFragmentos.get(i));
```

```
330             if ((clase == Categoria.class && reqCatName.contains(
                nuevaListaFragmentos.get(i))) || (clase == Cuenta.
                class && reqCueName.contains(nuevaListaFragmentos.get
                (i)))) {
331                 if (nuevaListaBundles.get(i) != null) {
332                     if ((clase == Categoria.class &&
                        nuevaListaBundles.get(i).getString(
                        ColCategoria.NOMBRE.toString()).equals(
                        antiguoValor)) || (clase == Cuenta.class &&
                        nuevaListaBundles.get(i).getString(ColCuenta.
                        NOMBRE.toString()).equals(antiguoValor))) {
333                         Log.i(TAG, "Descartando fragmento: " +
                            nuevaListaFragmentos.get(i));
334                     } else {
335                         if (listaFragmentos.size() == 0 ||
                            nuevaListaFragmentos.get(i) !=
                            listaFragmentos.get(listaFragmentos.size
                            () - 1)) {
336                             Log.i(TAG, "Volviendo a agregar
                                fragmento: " + nuevaListaFragmentos.
                                get(i));
337                             listaFragmentos.add(
                                nuevaListaFragmentos.get(i));
338                             listaBundles.add(nuevaListaBundles.
                                get(i));
339                         } else {
340                             Log.i(TAG, "Descartando fragmento: "
                                + nuevaListaFragmentos.get(i));
341                         }
342                     }
343                 } else {
344                     if (listaFragmentos.size() == 0 ||
                        nuevaListaFragmentos.get(i) !=
                        listaFragmentos.get(listaFragmentos.size() -
                        1)) {
345                         Log.i(TAG, "Volviendo a agregar
                            fragmento: " + nuevaListaFragmentos.get(i
                            ));
346                         listaFragmentos.add(nuevaListaFragmentos.
                            get(i));
347                         listaBundles.add(nuevaListaBundles.get(i
                            ));
348                     } else {
349                         Log.i(TAG, "Descartando fragmento: " +
                            nuevaListaFragmentos.get(i));
350                     }
351                 }
352             } else {
353                 if (listaFragmentos.size() == 0 ||
                    nuevaListaFragmentos.get(i) != listaFragmentos.
                    get(listaFragmentos.size() - 1)) {
354                     Log.i(TAG, "Volviendo a agregar fragmento: "
                        + nuevaListaFragmentos.get(i));
355                     listaFragmentos.add(nuevaListaFragmentos.get(
                        i));
```

```
356                         listaBundles.add(nuevaListaBundles.get(i));
357                     } else {
358                         Log.i(TAG, "Descartando fragmento: " +
                            nuevaListaFragmentos.get(i));
359                     }
360                 }
361             }
362         }
363     }
364
365     public void userLogIn(String llaveEncrip) {
366         Log.i(TAG, "El usuario ha iniciado sesión");
367         sesionIniciada = true;
368         this.llaveEncrip = llaveEncrip;
369         Log.i(TAG, "Llave Encriptación: " + llaveEncrip);
370         cambiarFragmento(new FragCuentas());
371         //invalidateOptionsMenu();
372     }
373
374     public void userLogOut() {
375         Log.i(TAG, "El usuario ha cerrado sesión");
376         sesionIniciada = false;
377         this.llaveEncrip = null;
378         if (menu != null) {
379             Log.i(TAG, "Cerrando el menú");
380             menu.close();
381         }
382         ParametroDAO parametroDAO = new ParametroDAO(
            getApplicationContext());
383         Parametro parametro = parametroDAO.seleccionarUno(
            NombreParametro.RESULTADO_HASH.toString());
384         if (parametro != null) {
385             if ("".equals(parametro.getValor())) {
386                 cambiarFragmento(new FragCrearLlaveMaestra());
387                 return;
388             }
389         }
390
391         cambiarFragmento(new FragInicioSesion());
392     }
393
394     //Se transforma el historial de fragmentos en una lista de strings
395     private ArrayList<String> listaFragmentoToStrings() {
396         ArrayList<String> resultado = new ArrayList<>();
397         for (Class clase : listaFragmentos) {
398             resultado.add(classToString(clase));
399         }
400         return resultado;
401     }
402
403     private String classToString(Class clase) {
404         return clase.getName();
405     }
406
```

```
407     //Se transforma el historial de bundles en una lista de strings
408     private ArrayList<String> listaBundlesToStrings() {
409         ArrayList<String> resultado = new ArrayList<>();
410         for (Bundle bundle : listaBundles) {
411             resultado.add(bundleToString(bundle));
412         }
413         return resultado;
414     }
415
416     private String bundleToString(Bundle bundle) {
417         if (bundle == null) {
418             return null;
419         }
420         if (bundle.getString(ColCategoria.NOMBRE.toString()) != null)
             {
421             return Categoria.class.getName() + ":" + bundle.getString
                (ColCategoria.NOMBRE.toString());
422         }
423         if (bundle.getString(ColCuenta.NOMBRE.toString()) != null) {
424             return Cuenta.class.getName() + ":" + bundle.getString(
                ColCuenta.NOMBRE.toString());
425         }
426         return null;
427     }
428
429     //Se carga la lista de strings en el historial de fragmentos
430     private void stringsToListaFragmentos(ArrayList<String> entrada) {
431         Log.i(TAG, entrada.toString());
432         listaFragmentos.clear();
433         for (String nombreFrag : entrada) {
434             listaFragmentos.add(stringToClass(nombreFrag));
435         }
436     }
437
438     private Class stringToClass(String nombre) {
439         if (FragAcercaDe.class.getName().equals(nombre)) {
440             return FragAcercaDe.class;
441         }
442         if (FragAgregarEditarCategoria.class.getName().equals(nombre
            )) {
443             return FragAgregarEditarCategoria.class;
444         }
445         if (FragAgregarEditarCuenta.class.getName().equals(nombre)) {
446             return FragAgregarEditarCuenta.class;
447         }
448         if (FragCambioLlaveMaestra.class.getName().equals(nombre)) {
449             return FragCambioLlaveMaestra.class;
450         }
451         if (FragCategorias.class.getName().equals(nombre)) {
452             return FragCategorias.class;
453         }
454         if (FragConfiguracion.class.getName().equals(nombre)) {
455             return FragConfiguracion.class;
456         }
457         if (FragCrearLlaveMaestra.class.getName().equals(nombre)) {
```

```
458             return FragCrearLlaveMaestra.class;
459         }
460         if (FragCuentas.class.getName().equals(nombre)) {
461             return FragCuentas.class;
462         }
463         if (FragDetalleCategoria.class.getName().equals(nombre)) {
464             return FragDetalleCategoria.class;
465         }
466         if (FragDetalleCuenta.class.getName().equals(nombre)) {
467             return FragDetalleCuenta.class;
468         }
469         if (FragExportar.class.getName().equals(nombre)) {
470             return FragExportar.class;
471         }
472         if (FragHistorialCuenta.class.getName().equals(nombre)) {
473             return FragHistorialCuenta.class;
474         }
475         if (FragInicioSesion.class.getName().equals(nombre)) {
476             return FragInicioSesion.class;
477         }
478         return null;
479     }
480
481     //Se carga la lista de strings en el historial de bundles
482     private void stringsToListaBundles(ArrayList<String> entrada) {
483         Log.i(TAG, entrada.toString());
484         listaBundles.clear();
485         for (String parametros : entrada) {
486             listaBundles.add(stringToBundle(parametros));
487         }
488     }
489
490     private Bundle stringToBundle(String datos) {
491         if (datos == null) {
492             return null;
493         }
494         int posSeparador = datos.indexOf(":");
495         String tipo = datos.substring(0, posSeparador);
496         String valor = datos.substring(posSeparador + 1);
497         if (Categoria.class.getName().equals(tipo)) {
498             Bundle nuevo = new Bundle();
499             nuevo.putString(ColCategoria.NOMBRE.toString(), valor);
500             return nuevo;
501         }
502         if (Cuenta.class.getName().equals(tipo)) {
503             Bundle nuevo = new Bundle();
504             nuevo.putString(ColCuenta.NOMBRE.toString(), valor);
505             return nuevo;
506         }
507         return null;
508     }
509
510     public boolean isSesionIniciada(){
511         return sesionIniciada;
512     }
513 }
```

FragAgregarEditarCuenta.java 1/14

```java
package cl.theroot.passbank.fragmento;

import android.app.AlertDialog;
import android.app.Fragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.ContextMenu;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SeekBar;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.Calendar;
import java.text.SimpleDateFormat;
import java.util.List;

import cl.theroot.passbank.ActividadPrincipal;
import cl.theroot.passbank.Cifrador;
import cl.theroot.passbank.CustomToast;
import cl.theroot.passbank.GeneradorContrasennas;
import cl.theroot.passbank.adaptador.AdapCategoriasCheckBox;
import cl.theroot.passbank.ExcepcionBancoContrasennas;
import cl.theroot.passbank.R;
import cl.theroot.passbank.dominio.Categoria;
import cl.theroot.passbank.dominio.CategoriaCuenta;
import cl.theroot.passbank.dominio.CategoriaSeleccionable;
import cl.theroot.passbank.dominio.Contrasenna;
import cl.theroot.passbank.dominio.Cuenta;
import cl.theroot.passbank.dominio.Parametro;
import cl.theroot.passbank.datos.CategoriaCuentaDAO;
import cl.theroot.passbank.datos.CategoriaDAO;
import cl.theroot.passbank.datos.ContrasennaDAO;
import cl.theroot.passbank.datos.CuentaDAO;
import cl.theroot.passbank.datos.ParametroDAO;
import cl.theroot.passbank.datos.nombres.ColCuenta;
import cl.theroot.passbank.datos.nombres.NombreParametro;

public class FragAgregarEditarCuenta extends Fragment implements View
.OnClickListener{
    private AdapCategoriasCheckBox adapter;
    private List<CategoriaSeleccionable> categories;

    private EditText ET_name;
```

FragAgregarEditarCuenta.java 2/14

```
55      private EditText ET_description;
56      private EditText ET_password;
57      private EditText ET_validez;
58      private ImageView IV_passVisibility;
59
60      private String oldName;
61      private Long oldPasswordID;
62
63      private String addEdit;
64      private Bundle bundle;
65
66      private ParametroDAO parametroDAO;
67      private CuentaDAO cuentaDAO;
68      private ContrasennaDAO contrasennaDAO;
69      private CategoriaCuentaDAO categoriaCuentaDAO;
70      private CategoriaDAO categoriaDAO;
71
72      private GeneradorContrasennas genContrasennas;
73      private byte modoGenerador = 0;
74
75      //Datos originales
76      private Boolean algunCambio = false;
77      private String nombreOriginal = "";
78      private String descripcionOriginal = "";
79      private String contrasennaOriginal = "";
80      private String validezOriginal = "";
81
82      @Override
83      public View onCreateView(LayoutInflater inflater, ViewGroup
        container, Bundle savedInstanceState) {
84          setHasOptionsMenu(true);
85          categoriaDAO = new CategoriaDAO(getActivity().
            getApplicationContext());
86          categoriaCuentaDAO = new CategoriaCuentaDAO(getActivity().
            getApplicationContext());
87          contrasennaDAO = new ContrasennaDAO(getActivity().
            getApplicationContext());
88          cuentaDAO = new CuentaDAO(getActivity().getApplicationContext
            ());
89          parametroDAO = new ParametroDAO(getActivity().
            getApplicationContext());
90
91          genContrasennas = new GeneradorContrasennas(getActivity().
            getApplicationContext());
92
93          View view = inflater.inflate(R.layout.
            fragmento_agregar_editar_cuenta, null);
94          TextView TV_titule = view.findViewById(R.id.TV_titule);
95          TextView TV_subTitulo = view.findViewById(R.id.TV_subTitulo);
96          ET_name = view.findViewById(R.id.ET_name);
97          ET_description = view.findViewById(R.id.ET_description);
98          ET_password = view.findViewById(R.id.ET_password);
99          ET_validez = view.findViewById(R.id.ET_validez);
100         IV_passVisibility = view.findViewById(R.id.IV_passVisibility);
101         IV_passVisibility.setOnClickListener(this);
```

```
102         ImageView IV_passGenerator = view.findViewById(R.id.
            IV_passGenerator);
103         IV_passGenerator.setOnClickListener(this);
104         registerForContextMenu(IV_passGenerator);
105
106         ListView listView = view.findViewById(R.id.
            listview_categories_checkboxs);
107         listView.setOnTouchListener(new View.OnTouchListener() {
108             @Override
109             public boolean onTouch(View view, MotionEvent motionEvent
                ) {
110                 view.getParent().requestDisallowInterceptTouchEvent(
                    true);
111                 return false;
112             }
113         });
114
115
116         final SeekBar SB_validez = view.findViewById(R.id.SB_validez);
117         SB_validez.setOnSeekBarChangeListener(new SeekBar.
            OnSeekBarChangeListener() {
118             @Override
119             public void onProgressChanged(SeekBar seekBar, int
                progress, boolean fromUser) {
120                 if (fromUser) {
121                     ET_validez.setText(String.valueOf(progress));
122                 }
123             }
124
125             @Override
126             public void onStartTrackingTouch(SeekBar seekBar) {
127
128             }
129
130             @Override
131             public void onStopTrackingTouch(SeekBar seekBar) {
132
133             }
134         });
135
136         ET_validez.addTextChangedListener(new TextWatcher() {
137             @Override
138             public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
139
140             }
141
142             @Override
143             public void onTextChanged(CharSequence s, int start, int
                before, int count) {
144
145             }
146
147             @Override
148             public void afterTextChanged(Editable s) {
```

```
149             try {
150                 Integer numero = Integer.parseInt(ET_validez.
                    getText().toString());
151                 if (numero > SB_validez.getMax()) {
152                     SB_validez.setProgress(SB_validez.getMax());
153                 } else {
154                     if (numero < 0) {
155                         SB_validez.setProgress(0);
156                     } else {
157                         SB_validez.setProgress(numero);
158                     }
159                 }
160             } catch(NumberFormatException ex) {
161                 SB_validez.setProgress(0);
162             }
163         }
164     });
165
166     bundle = this.getArguments();
167     oldName = null;
168     oldPasswordID = null;
169
170     TV_titule.setText("Crear Cuenta");
171     addEdit = "ADD";
172     Parametro parValDefecto = parametroDAO.seleccionarUno(
    NombreParametro.VALIDEZ_DEFECTO.toString());
173     if (parValDefecto != null) {
174         ET_validez.setText(parValDefecto.getValor());
175     } else {
176         ET_validez.setText('0');
177     }
178
179     if (bundle != null) {
180         oldName = bundle.getString(ColCuenta.NOMBRE.toString());
181         if (oldName != null) {
182             Cuenta cuenta = cuentaDAO.seleccionarUna(oldName);
183             if (cuenta != null) {
184                 TV_titule.setText("Editar Cuenta");
185                 addEdit = "EDIT";
186                 ET_name.setText(cuenta.getNombre());
187                 ET_description.setText(cuenta.getDescripcion());
188                 ET_validez.setText(String.valueOf(cuenta.
                    getValidez()));
189
190                 Contrasenna contrasenna = contrasennaDAO.
                    seleccionarUltimaPorCuenta(cuenta.getNombre());
191                 if (contrasenna != null) {
192                     oldPasswordID = contrasenna.getId();
193                     ET_password.setText(Cifrador.desencriptar(
                        contrasenna.getValor(), ActividadPrincipal.
                        llaveEncrip));
194                 } else {
195                     oldPasswordID = null;
196                 }
197             } else {
```

```
198                     oldName = null;
199                 }
200             }
201         }
202
203         fillCategoriesInfo(oldName);
204         adapter = new AdapCategoriasCheckBox(getActivity(),
            categories, this);
205         listView.setAdapter(adapter);
206         setListViewHeightBasedOnChildren(listView);
207
208         if (categories.isEmpty()) {
209             TV_subTitulo.setVisibility(View.INVISIBLE);
210         }
211
212         //Setear el modo del generador por defecto
213         Parametro parametro = parametroDAO.seleccionarUno(
            NombreParametro.ULTIMO_MODO_GENERADOR.toString());
214         if (parametro != null) {
215             modoGenerador = Byte.parseByte(parametro.getValor());
216         }
217
218         //Setear datos para habilitar/deshabilitar el botón guardar
219         algunCambio = false;
220         nombreOriginal = ET_name.getText().toString();
221         ET_name.addTextChangedListener(new TextWatcher() {
222             @Override
223             public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
224
225             }
226
227             @Override
228             public void onTextChanged(CharSequence s, int start, int
                before, int count) {
229
230             }
231
232             @Override
233             public void afterTextChanged(Editable s) {
234                 checkearCambios();
235             }
236         });
237         descripcionOriginal = ET_description.getText().toString();
238         ET_description.addTextChangedListener(new TextWatcher() {
239             @Override
240             public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
241
242             }
243
244             @Override
245             public void onTextChanged(CharSequence s, int start, int
                before, int count) {
246
```

```
247             }
248
249             @Override
250             public void afterTextChanged(Editable s) {
251                 checkearCambios();
252             }
253         });
254         contrasennaOriginal = ET_password.getText().toString();
255         ET_password.addTextChangedListener(new TextWatcher() {
256             @Override
257             public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
258
259             }
260
261             @Override
262             public void onTextChanged(CharSequence s, int start, int
                before, int count) {
263
264             }
265
266             @Override
267             public void afterTextChanged(Editable s) {
268                 checkearCambios();
269             }
270         });
271         validezOriginal = ET_validez.getText().toString();
272         ET_validez.addTextChangedListener(new TextWatcher() {
273             @Override
274             public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
275
276             }
277
278             @Override
279             public void onTextChanged(CharSequence s, int start, int
                before, int count) {
280
281             }
282
283             @Override
284             public void afterTextChanged(Editable s) {
285                 checkearCambios();
286             }
287         });
288
289         return view;
290     }
291
292     //Creación del submenu del fragmento
293     @Override
294     public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
         {
295         inflater.inflate(R.menu.sub_menu_agregar_editar_cuenta, menu);
296         super.onCreateOptionsMenu(menu, inflater);
```

FragAgregarEditarCuenta.java 7/14

```
297     }
298
299     @Override
300     public void onPrepareOptionsMenu(Menu menu){
301         menu.findItem(R.id.sub_menu_add_edit_account_save).setEnabled
            (algunCambio);
302         super.onPrepareOptionsMenu(menu);
303     }
304
305     //Creación de la funcionalidad del submenu del fragmento
306     @Override
307     public boolean onOptionsItemSelected(MenuItem item) {
308         if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
             {
309             return false;
310         }
311         try {
312             switch (item.getItemId()) {
313                 case R.id.sub_menu_add_edit_account_back:
314                     getActivity().onBackPressed();
315                     return true;
316
317                 case R.id.sub_menu_add_edit_account_save:
318                     String name = ET_name.getText().toString().trim();
319                     String description = ET_description.getText().
                        toString().trim();
320                     String password = ET_password.getText().toString
                        ();
321                     String strValidez = ET_validez.getText().toString
                        ().trim();
322                     Integer validez;
323
324                     if (description.equals("")) {
325                         description = null;
326                     }
327
328                     // Revisar si el nombre y contraseña están vacíos.
329                     if (name.equals("")) {
330                         throw new ExcepcionBancoContrasennas("Error
                            - Campo Vacío", "Toda cuenta requiere de un
                            nombre, favor de rellenar el campo Nombre");
331                     }
332                     if (password.equals("")) {
333                         throw new ExcepcionBancoContrasennas("Error
                            - Campo Vacío", "Toda cuenta requiere de una
                            contraseña, favor de rellenar el campo
                            Contraseña");
334                     }
335
336                     try {
337                         validez = Integer.parseInt(strValidez);
338                     } catch (NumberFormatException ex) {
339                         throw new ExcepcionBancoContrasennas("Error
                            - Formato Inválido", "La validez debe ser un
                            número entero");
```

```
340                 }
341
342                 if (validez < 0) {
343                     throw new ExcepcionBancoContrasennas("Error
                        - Formato Inválido", "La validez debe ser un
                        número mayor a 0");
344                 }
345
346                 //Revisar si ya existe un cuenta con el mismo
                    nombre
347                 Cuenta cuentaIdentica = cuentaDAO.seleccionarUna(
                    name);
348                 if (cuentaIdentica != null && !name.equals(
                    oldName)) {
349                     throw new ExcepcionBancoContrasennas("Error
                        - Cuenta Existente", "Ya existe una cuenta
                        con el nombre establecido");
350                 }
351
352                 Cuenta cuenta = new Cuenta(name, description,
                    validez);
353                 //Se actualiza o inserta la cuenta deseada
354                 String nombreAntiguo = null;
355                 if (addEdit.equals("ADD")) {
356                     if (cuentaDAO.insertarUna(cuenta) == -1) {
357                         throw new ExcepcionBancoContrasennas(
                            "Error - Cuenta No Creada", "Hubo un
                            error con la base de datos, y su cuenta
                            no fue creada.");
358                     }
359                 } else {
360                     if (cuentaDAO.actualizarUna(oldName, cuenta)
                        == 0) {
361                         throw new ExcepcionBancoContrasennas(
                            "Error - Cuenta No Modificada", "Hubo un
                            error con la base de datos, y su cuenta
                            no fue modificada.");
362                     } else {
363                         nombreAntiguo = oldName;
364                     }
365                 }
366                 oldName = cuenta.getNombre();
367
368                 //Se obtiene la fecha de creación de la contraseña
369                 Calendar calendar = Calendar.getInstance();
370                 SimpleDateFormat simpleDateFormat = new
                    SimpleDateFormat("yyyy/MM/dd");
371                 String formattedDate = simpleDateFormat.format(
                    calendar.getTime());
372
373                 if (oldPasswordID != null) {
374                     //Se obtiene la contraseña actual de la
                        cuenta para comparar si es igual a la nueva
375                     Contrasenna contAntigua = contrasennaDAO.
                        seleccionarUna(oldPasswordID);
```

FragAgregarEditarCuenta.java 9/14

```
376                 if (contAntigua != null) {
377                     if (!Cifrador.desencriptar(contAntigua.
                        getValor(), ActividadPrincipal.
                        llaveEncrip).equals(password)) {
378                         Contrasenna contrasenna = new
                            Contrasenna(null, oldName, Cifrador.
                            encriptar(password,
                            ActividadPrincipal.llaveEncrip),
                            formattedDate);
379                         oldPasswordID = contrasennaDAO.
                            insertarUna(contrasenna);
380                     }
381                 } else {
382                     Contrasenna contrasenna = new Contrasenna
                        (null, oldName, Cifrador.encriptar(
                        password, ActividadPrincipal.llaveEncrip
                        ), formattedDate);
383                     oldPasswordID = contrasennaDAO.
                        insertarUna(contrasenna);
384                 }
385             } else {
386                 Contrasenna contrasenna = new Contrasenna(
                    null, oldName, Cifrador.encriptar(password,
                    ActividadPrincipal.llaveEncrip),
                    formattedDate);
387                 oldPasswordID = contrasennaDAO.insertarUna(
                    contrasenna);
388             }
389
390             //Ya creada/actualizada la cuenta y contraseña,
                se deben actualizar las relaciones entre
                categorias y cuentas.
391             for (CategoriaSeleccionable categoria : adapter.
                getCategorias()) {
392                 //Checkear si existe la relación
393                 CategoriaCuenta categoriaCuenta =
                    categoriaCuentaDAO.seleccionarUna(categoria.
                    getNombre(), oldName);
394                 if (categoria.isSeleccionado()) {
395                     //Si no existe, la creamos.
396                     if (categoriaCuenta == null) {
397                         categoriaCuentaDAO.insertarUna(new
                            CategoriaCuenta(categoria.getNombre
                            (), oldName, 0));
398                     }
399                 } else {
400                     //Si existe, la eliminamos.
401                     if (categoriaCuenta != null) {
402                         categoriaCuentaDAO.eliminarUna(
                            categoria.getNombre(), oldName);
403                     }
404                 }
405             }
406
407             if (addEdit.equals("ADD")) {
```

```
408                         new CustomToast(getActivity().
                            getApplicationContext(), "Su cuenta fue
                            creada exitosamente.");
409                     } else {
410                         new CustomToast(getActivity().
                            getApplicationContext(), "Su cuenta fue
                            modificada exitosamente.");
411                     }
412
413                     ((ActividadPrincipal) getActivity()).
                        cambiarFragmento(new FragCuentas());
414                     if (nombreAntiguo != null) {
415                         ((ActividadPrincipal) getActivity()).
                            actualizarBundles(Cuenta.class, nombreAntiguo
                            , cuenta.getNombre());
416                     }
417                     return true;
418                 default:
419                     return super.onOptionsItemSelected(item);
420             }
421         } catch(ExcepcionBancoContrasennas ex) {
422             AlertDialog alertDialog = new AlertDialog.Builder(
                getActivity()).create();
423             alertDialog.setTitle(ex.getTitulo());
424             alertDialog.setMessage(ex.getMensaje());
425             alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
                new DialogInterface.OnClickListener() {
426                 public void onClick(DialogInterface dialog, int which
                    ) {
427                     dialog.dismiss();
428                 }
429             });
430             alertDialog.show();
431             return true;
432         }
433     }
434
435     @Override
436     public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenu.ContextMenuInfo menuInfo) {
437         super.onCreateContextMenu(menu, v, menuInfo);
438         MenuInflater inflater = getActivity().getMenuInflater();
439         inflater.inflate(R.menu.cont_menu_agregar_editar_cuenta, menu
            );
440         switch(modoGenerador) {
441             case 0:
442                 menu.findItem(R.id.
                    cont_menu_agregar_editar_cuenta_caracteres).
                    setChecked(true);
443                 break;
444             case 1:
445                 menu.findItem(R.id.
                    cont_menu_agregar_editar_cuenta_palabras).setChecked(
                    true);
446                 break;
```

```
447         }
448     }
449
450     //Creación de la funcionalidad del menu contextual
451     @Override
452     public boolean onContextItemSelected(MenuItem item) {
453         if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
                {
454             return false;
455         }
456
457         switch (item.getItemId()) {
458             case R.id.cont_menu_agregar_editar_cuenta_caracteres:
459                 item.setChecked(true);
460                 modoGenerador = 0;
461
462                 //Para mantener abierto el context menu al hacer click
463                 item.setShowAsAction(MenuItem.
                    SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW);
464                 item.setActionView(new View(getActivity().
                    getApplicationContext()));
465                 item.setOnActionExpandListener(new MenuItem.
                    OnActionExpandListener() {
466                     @Override
467                     public boolean onMenuItemActionExpand(MenuItem
                        item) {
468                         return false;
469                     }
470
471                     @Override
472                     public boolean onMenuItemActionCollapse(MenuItem
                        item) {
473                         return false;
474                     }
475                 });
476
477                 return false;
478             case R.id.cont_menu_agregar_editar_cuenta_palabras:
479                 item.setChecked(true);
480                 modoGenerador = 1;
481
482                 //Para mantener abierto el context menu al hacer click
483                 item.setShowAsAction(MenuItem.
                    SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW);
484                 item.setActionView(new View(getActivity().
                    getApplicationContext()));
485                 item.setOnActionExpandListener(new MenuItem.
                    OnActionExpandListener() {
486                     @Override
487                     public boolean onMenuItemActionExpand(MenuItem
                        item) {
488                         return false;
489                     }
490
491                     @Override
```

```
492                     public boolean onMenuItemActionCollapse(MenuItem
                        item) {
493                         return false;
494                     }
495                 });
496
497                 return false;
498             default:
499                 return super.onContextItemSelected(item);
500         }
501     }
502
503     //Funcionalidad del botón para hacer visible/invisible la
        contraseña
504     @Override
505     public void onClick(View view) {
506         switch (view.getId()) {
507             case R.id.IV_passVisibility:
508                 if (ET_password.getInputType() == 0x00000081) {
509                     int pointerPos = ET_password.getSelectionStart();
510                     IV_passVisibility.setImageResource(R.drawable.
                        ic_visibility_off_white_24dp);
511                     ET_password.setInputType(0x00080001);
512                     ET_password.setSelection(pointerPos);
513                 }
514                 else {
515                     int pointerPos = ET_password.getSelectionStart();
516                     IV_passVisibility.setImageResource(R.drawable.
                        ic_visibility_white_24dp);
517                     ET_password.setInputType(0x00000081);
518                     ET_password.setSelection(pointerPos);
519                 }
520                 break;
521             case R.id.IV_passGenerator:
522                 parametroDAO.actualizarUna(new Parametro(
                    NombreParametro.ULTIMO_MODO_GENERADOR.toString(),
                    String.valueOf(modoGenerador), null));
523                 if (modoGenerador == 0) {
524                     ET_password.setText(genContrasennas.generar(true
                        ));
525                 } else {
526                     ET_password.setText(genContrasennas.generar(false
                        ));
527                 }
528                 break;
529         }
530     }
531
532     private void fillCategoriesInfo(String nombreCuenta) {
533         categories = new ArrayList<>();
534
535         for (Categoria categoria : categoriaDAO.seleccionarTodas()) {
536             CategoriaSeleccionable categoriaSeleccionable = new
                CategoriaSeleccionable(categoria.getNombre(), categoria.
                getPosicion(), false);
```

```
537             if (nombreCuenta != null) {
538                 CategoriaCuenta categoriaCuenta = categoriaCuentaDAO.
                    seleccionarUna(categoria.getNombre(), nombreCuenta);
539                 if (categoriaCuenta != null) {
540                     categoriaSeleccionable.setSeleccionado(true);
541                 }
542             }
543             categories.add(categoriaSeleccionable);
544         }
545     }
546 
547     public static void setListViewHeightBasedOnChildren(ListView
    listView) {
548         ListAdapter listAdapter = listView.getAdapter();
549         if (listAdapter == null) {
550             return;
551         }
552         int desiredWidth = View.MeasureSpec.makeMeasureSpec(listView.
            getWidth(), View.MeasureSpec.UNSPECIFIED);
553         int totalHeight = 0;
554         View view = null;
555         for (int i = 0; i < listAdapter.getCount(); i++) {
556             view = listAdapter.getView(i, view, listView);
557             if (i == 0)
558                 view.setLayoutParams(new ViewGroup.LayoutParams(
                    desiredWidth, ViewGroup.LayoutParams.WRAP_CONTENT));
559 
560             view.measure(desiredWidth, View.MeasureSpec.UNSPECIFIED);
561             totalHeight += view.getMeasuredHeight();
562         }
563         ViewGroup.LayoutParams params = listView.getLayoutParams();
564         params.height = totalHeight + (listView.getDividerHeight() *
            (listAdapter.getCount() - 1));
565         listView.setLayoutParams(params);
566     }
567 
568     public void checkearCambios() {
569         algunCambio = false;
570         if (!ET_name.getText().toString().trim().equals(
            nombreOriginal)) {
571             algunCambio = true;
572             getActivity().invalidateOptionsMenu();
573             return;
574         }
575         if (!ET_description.getText().toString().trim().equals(
            descripcionOriginal)) {
576             algunCambio = true;
577             getActivity().invalidateOptionsMenu();
578             return;
579         }
580         if (!ET_password.getText().toString().equals(
            contrasennaOriginal)) {
581             algunCambio = true;
582             getActivity().invalidateOptionsMenu();
583             return;
```

```
584             }
585             if (!ET_validez.getText().toString().trim().equals(
                validezOriginal)) {
586                 algunCambio = true;
587                 getActivity().invalidateOptionsMenu();
588                 return;
589             }
590             if (adapter.checkearCambios()) {
591                 algunCambio = true;
592                 getActivity().invalidateOptionsMenu();
593                 return;
594             }
595             getActivity().invalidateOptionsMenu();
596         }
597     }
598
```

FragCambioLlaveMaestra.java 1/5

```java
package cl.theroot.passbank.fragmento;


import android.app.AlertDialog;
import android.app.Fragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;

import cl.theroot.passbank.ActividadPrincipal;
import cl.theroot.passbank.Cifrador;
import cl.theroot.passbank.CustomToast;
import cl.theroot.passbank.ExcepcionBancoContrasennas;
import cl.theroot.passbank.R;
import cl.theroot.passbank.dominio.Contrasenna;
import cl.theroot.passbank.dominio.Parametro;
import cl.theroot.passbank.datos.ContrasennaDAO;
import cl.theroot.passbank.datos.ParametroDAO;
import cl.theroot.passbank.datos.nombres.NombreParametro;

/**
 * A simple {@link Fragment} subclass.
 */
public class FragCambioLlaveMaestra extends Fragment{
    private static final String TAG = "BdC-FragCambioLlaveMaestra";
    private EditText ET_oldPassword;
    private EditText ET_newPassword;
    private EditText ET_newRepPassword;
    private ParametroDAO parametroDAO;
    private boolean habilitarCambios = false;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
        setHasOptionsMenu(true);
        View view = inflater.inflate(R.layout.
        fragmento_cambio_llave_maestra, null);
        ET_oldPassword = view.findViewById(R.id.ET_oldPassword);
        ET_newPassword = view.findViewById(R.id.ET_newPassword);
        ET_newRepPassword = view.findViewById(R.id.ET_newRepPassword);
        ET_oldPassword.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start,
            int count, int after) {

            }

```

FragCambioLlaveMaestra.java 2/5

```
53              @Override
54              public void onTextChanged(CharSequence s, int start, int
                before, int count) {
55
56              }
57
58              @Override
59              public void afterTextChanged(Editable s) {
60                  habilitarCambios();
61              }
62          });
63          ET_newPassword.addTextChangedListener(new TextWatcher() {
64              @Override
65              public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
66
67              }
68
69              @Override
70              public void onTextChanged(CharSequence s, int start, int
                before, int count) {
71
72              }
73
74              @Override
75              public void afterTextChanged(Editable s) {
76                  habilitarCambios();
77              }
78          });
79          ET_newRepPassword.addTextChangedListener(new TextWatcher() {
80              @Override
81              public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
82
83              }
84
85              @Override
86              public void onTextChanged(CharSequence s, int start, int
                before, int count) {
87
88              }
89
90              @Override
91              public void afterTextChanged(Editable s) {
92                  habilitarCambios();
93              }
94          });
95          parametroDAO = new ParametroDAO(getActivity().
            getApplicationContext());
96          return view;
97      }
98
99      //Creación del submenú del fragmento
100     @Override
```

FragCambioLlaveMaestra.java 3/5

```
101     public void onCreateOptionsMenu(Menu menu, MenuInflater
        menuInflater) {
102         menuInflater.inflate(R.menu.sub_menu_cambio_llave_maestra,
            menu);
103         super.onCreateOptionsMenu(menu, menuInflater);
104     }
105
106     @Override
107     public void onPrepareOptionsMenu(Menu menu){
108         menu.findItem(R.id.sub_menu_cambio_llave_maestra_guardar).
            setEnabled(habilitarCambios);
109         super.onPrepareOptionsMenu(menu);
110     }
111
112     @Override
113     public boolean onOptionsItemSelected(MenuItem menuItem) {
114         if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
             {
115             return false;
116         }
117         switch(menuItem.getItemId()) {
118             case R.id.sub_menu_cambio_llave_maestra_volver:
119                 getActivity().onBackPressed();
120                 return true;
121             case R.id.sub_menu_cambio_llave_maestra_guardar:
122                 try {
123                     //Procesar cambio de contraseña
124                     String contVieja = ET_oldPassword.getText().
                        toString();
125                     String contNueva = ET_newPassword.getText().
                        toString();
126                     String contNuevaConf = ET_newRepPassword.getText
                        ().toString();
127
128                     //Validar contraseña vieja
129                     if (contVieja.isEmpty()) {
130                         throw new ExcepcionBancoContrasennas("Error
                            - Llave Maestra Requerida", "Se requiere el
                            ingreso de su Llave Maestra Actual para
                            realizar el cambio");
131                     }
132                     Parametro parSalt = parametroDAO.seleccionarUno(
                        NombreParametro.SAL_HASH.toString());
133                     Parametro parHash = parametroDAO.seleccionarUno(
                        NombreParametro.RESULTADO_HASH.toString());
134                     String[] saltYHashObt = Cifrador.genHashedPass(
                        contVieja, parSalt.getValor());
135                     if (!saltYHashObt[1].equals(parHash.getValor())) {
136                         throw new ExcepcionBancoContrasennas("Error
                            - Llave Maestra Incorrecta", "La Llave
                            Maestra Actual ingresada no es correcta,
                            favor intentar de nuevo");
137                     }
138
```

```
139                 //Validar contraseña nueva
140                 if (contNueva.isEmpty()) {
141                     throw new ExcepcionBancoContrasennas("Error
                        - Campo Vacío", "Para realizar el cambio, se
                        requiere el ingreso de su Nueva Llave
                        Maestra");
142                 }
143                 if (contNueva.length() < Cifrador.
                    LARGO_MINIMO_LLAVE_MAESTRA) {
144                     throw new ExcepcionBancoContrasennas("Error
                        - Llave Muy Corta", "La Nueva Llave Maestra
                        elegida es muy corta, debería tener al menos
                        " + Cifrador.LARGO_MINIMO_LLAVE_MAESTRA + "
                        caracteres");
145                 }
146                 if (!contNuevaConf.equals(contNueva)) {
147                     throw new ExcepcionBancoContrasennas("Error
                        - No Coincidencia", "No coinciden las Nuevas
                        Llaves Maestras ingresadas, favor reingresar
                        los datos");
148                 }
149
150                 //Validar que la contraseña nueva sea distinta a
                    la anterior
151                 if (contNueva.equals(contVieja)) {
152                     throw new ExcepcionBancoContrasennas("Error
                        - Llave Ya Utilizada", "La Nueva Llave
                        Maestra debe ser distinta a la Llave Maestra
                        Actual");
153                 }
154
155                 //Crear Hash y LlaveEncriptacion para la nueva
                    llave maestra
156                 String[] saltYHash = Cifrador.genHashedPass(
                    contNueva, null);
157                 String[] saltYHashEncr = Cifrador.genHashedPass(
                    contNueva, null);
158                 //Reencriptar, configurar base de datos, y
                    configurar ActividadPrincipal
159                 reencriptarContrasennas(ActividadPrincipal.
                    llaveEncrip, saltYHashEncr[1]);
160                 parSalt = new Parametro(NombreParametro.SAL_HASH.
                    toString(), saltYHash[0], null);
161                 parHash = new Parametro(NombreParametro.
                    RESULTADO_HASH.toString(), saltYHash[1], null);
162                 Parametro parSaltEncr = new Parametro(
                    NombreParametro.SAL_ENCRIPTACION.toString(),
                    saltYHashEncr[0], null);
163                 if (parametroDAO.actualizarUna(parSalt) > 0) {
164                     if (parametroDAO.actualizarUna(parHash) > 0) {
165                         if (parametroDAO.actualizarUna(
                            parSaltEncr) > 0) {
166                             ActividadPrincipal.llaveEncrip =
                                saltYHashEncr[1];
```

```
167                                     new CustomToast(getActivity().
                                        getApplicationContext(), "Su Llave
                                        Maestra fue actualizada exitosamente"
                                        );
168                                     ((ActividadPrincipal) getActivity()).
                                        cambiarFragmento(new
                                        FragConfiguracion());
169                                 }
170                             }
171                         }
172                     } catch(ExcepcionBancoContrasennas ex) {
173                         AlertDialog alertDialog = new AlertDialog.Builder
                        (getActivity()).create();
174                         alertDialog.setTitle(ex.getTitulo());
175                         alertDialog.setMessage(ex.getMensaje());
176                         alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL,
                         "OK", new DialogInterface.OnClickListener() {
177                             public void onClick(DialogInterface dialog,
                            int which) {
178                                 dialog.dismiss();
179                             }
180                         });
181                         alertDialog.show();
182                     }
183                     return true;
184                 default:
185                     return super.onOptionsItemSelected(menuItem);
186             }
187         }
188
189         private void reencriptarContrasennas(String llaveEncrVieja,
        String llaveEncrNueva) {
190             ContrasennaDAO contrasennaDAO = new ContrasennaDAO(
                getActivity().getApplicationContext());
191             for (Contrasenna contrasenna : contrasennaDAO.
                seleccionarTodas()) {
192                 String valorDesencriptado = Cifrador.desencriptar(
                    contrasenna.getValor(), llaveEncrVieja);
193                 String valorEncriptado = Cifrador.encriptar(
                    valorDesencriptado, llaveEncrNueva);
194                 contrasenna.setValor(valorEncriptado);
195                 contrasennaDAO.actualizarUna(contrasenna);
196             }
197         }
198
199         private void habilitarCambios() {
200             habilitarCambios = true;
201             if (ET_oldPassword.getText().toString().isEmpty()) {
202                 habilitarCambios = false;
203             }
204             if (ET_newPassword.getText().toString().isEmpty()) {
205                 habilitarCambios = false;
206             }
207             if (ET_newRepPassword.getText().toString().isEmpty()) {
208                 habilitarCambios = false;
209             }
210             getActivity().invalidateOptionsMenu();
211         }
212     }
```

FragConfiguracion.java 1/4

```java
package cl.theroot.passbank.fragmento;


import android.app.AlertDialog;
import android.app.Fragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import cl.theroot.passbank.ActividadPrincipal;
import cl.theroot.passbank.adaptador.AdapParametros;
import cl.theroot.passbank.CustomToast;
import cl.theroot.passbank.R;
import cl.theroot.passbank.dominio.Parametro;
import cl.theroot.passbank.dominio.ParametroSeleccionable;
import cl.theroot.passbank.datos.CategoriaDAO;
import cl.theroot.passbank.datos.ParametroDAO;
import cl.theroot.passbank.datos.nombres.NombreParametro;


public class FragConfiguracion extends Fragment {
    private static final String TAG = "BdC-FragConfiguracion";
    private AdapParametros adapter;
    private List<ParametroSeleccionable> parametros;

    public static final String[] PARAMETROS_NUMERICOS = {
            NombreParametro.CANT_PALABRAS_GENERADOR.toString(),
            NombreParametro.CANT_CARACTERES_GENERADOR.toString(),
            NombreParametro.VALIDEZ_DEFECTO.toString()
    };

    public static final String[] PARAMETROS_NO_TRIMEABLES = {
            NombreParametro.SEPARADOR_GENERADOR.toString(),
            NombreParametro.COMPOSICION_GENERADOR.toString()
    };

    public static final String[] PARAMETROS_NO_REPETIBLES = {
            NombreParametro.COMPOSICION_GENERADOR.toString()
    };

    private boolean parametrosCambiados = false;

    private CategoriaDAO categoriaDAO;
    private ParametroDAO parametroDAO;
```

FragConfiguracion.java 2/4

```
56
57      @Override
58      public View onCreateView(LayoutInflater inflater, ViewGroup
        container, Bundle savedInstanceState) {
59          setHasOptionsMenu(true);
60          parametroDAO = new ParametroDAO(getActivity().
            getApplicationContext());
61          categoriaDAO = new CategoriaDAO(getActivity().
            getApplicationContext());
62
63          View view = inflater.inflate(R.layout.fragmento_configuracion
            , container, false);
64
65          ListView listView = view.findViewById(R.id.listConfiguration);
66
67          parametros = new ArrayList<>();
68          for (Parametro parametro : parametroDAO.seleccionarVisibles
            ()) {
69              parametros.add(new ParametroSeleccionable(parametro.
                getNombre(), parametro.getValor(), parametro.getPosicion
                (), false));
70          }
71          adapter = new AdapParametros(getActivity(), parametros, this);
72          listView.setAdapter(adapter);
73
74          return view;
75      }
76
77      //Creación del submenu del fragmento
78      @Override
79      public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
         {
80          Log.d(TAG, "onCreateOptionsMenu executed!");
81          inflater.inflate(R.menu.sub_menu_configuracion, menu);
82          super.onCreateOptionsMenu(menu, inflater);
83      }
84
85      @Override
86      public void onPrepareOptionsMenu(Menu menu){
87          Log.d(TAG, "onPrepareOptionsMenu executed!");
88          menu.findItem(R.id.sub_menu_configuration_save).setEnabled(
            parametrosCambiados);
89          super.onPrepareOptionsMenu(menu);
90      }
91
92      //Creación de la funcionalidad del submenu del fragmento
93      @Override
94      public boolean onOptionsItemSelected(MenuItem item) {
95          if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
             {
96              return false;
97          }
98          switch (item.getItemId()) {
99              case R.id.sub_menu_configuration_change_master_key:
```

```
100                 return ((ActividadPrincipal) getActivity()).
                    cambiarFragmento(new FragCambioLlaveMaestra());
101             case R.id.sub_menu_configuration_export_database:
102                 return ((ActividadPrincipal) getActivity()).
                    cambiarFragmento(new FragExportar());
103             case R.id.sub_menu_configuration_save:
104                 if (!parametrosCambiados) {
105                     new CustomToast(getActivity().
                        getApplicationContext(), "¡Acción Cancelada!\nNo
                        hay cambios que se deban almacenar.");
106                     return true;
107                 }
108                 parametros = adapter.getParametros();
109                 String salida = parametrosValidos(parametros);
110                 if (salida == null) {
111                     AlertDialog alertDialog = new AlertDialog.Builder
                        (getActivity()).create();
112                     alertDialog.setTitle("Configuración Actualizada");
113                     alertDialog.setMessage("Los cambios ejecutados
                        fueron guardados exitosamente.");
114                     alertDialog.setButton(AlertDialog.BUTTON_POSITIVE
                        , "OK", new DialogInterface.OnClickListener() {
115                         @Override
116                         public void onClick(DialogInterface dialog,
                            int which) {
117                             dialog.dismiss();
118                         }
119                     });
120                     alertDialog.show();
121
122                     for (Parametro parametro : parametros) {
123                         parametroDAO.actualizarUna(parametro);
124                     }
125
126                     parametros = new ArrayList<>();
127                     for (Parametro parametro : parametroDAO.
                        seleccionarVisibles()) {
128                         parametros.add(new ParametroSeleccionable(
                            parametro.getNombre(), parametro.getValor(),
                            parametro.getPosicion(), false));
129                     }
130                     adapter.updateParametros(parametros);
131                     return true;
132                 } else {
133                     AlertDialog alertDialog = new AlertDialog.Builder
                        (getActivity()).create();
134                     alertDialog.setTitle("Error, Datos Inválidos");
135                     alertDialog.setMessage(salida);
136                     alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL,
                         "OK", new DialogInterface.OnClickListener() {
137                         @Override
138                         public void onClick(DialogInterface dialog,
                            int which) {
139                             dialog.dismiss();
140                         }
```

```
141                     });
142                     alertDialog.show();
143                 }
144                 return super.onOptionsItemSelected(item);
145             default:
146                 return super.onOptionsItemSelected(item);
147         }
148     }
149
150     private String parametrosValidos(List<ParametroSeleccionable>
        parametros) {
151         for (Parametro parametro : parametros){
152             if (parametro.getValor() == null) {
153                 return parametro.getNombre() + "\nNo puede ser nulo.";
154             }
155
156             if (parametro.getValor().length() == 0) {
157                 return parametro.getNombre() + "\nNo puede estar
                    vacío.";
158             }
159
160             if (!Arrays.asList(FragConfiguracion.
                PARAMETROS_NO_TRIMEABLES).contains(parametro.getNombre
                ())) {
161                 if (parametro.getValor().trim().isEmpty()) {
162                     return parametro.getNombre() + ":\nNo puede
                        estar vacío.";
163                 }
164             }
165
166             if (Arrays.asList(PARAMETROS_NUMERICOS).contains(
                parametro.getNombre())) {
167                 try {
168                     Integer.parseInt(parametro.getValor());
169                 } catch(NumberFormatException ex) {
170                     return parametro.getNombre() + "\nDebe ser
                        numérico.";
171                 }
172             }
173
174             if (parametro.getNombre().equals(NombreParametro.
                NOMBRE_CATEGORIA_COMPLETA.toString())) {
175                 if (categoriaDAO.seleccionarUna(parametro.getValor())
                     != null) {
176                     return parametro.getNombre() + "\nDebe tener un
                        nombre distinto al resto de categorías.";
177                 }
178             }
179         }
180         return null;
181     }
182
183
184     public void habilitarCambios(Boolean estado) {
185         if (parametrosCambiados != estado) {
186             parametrosCambiados = estado;
187             getActivity().invalidateOptionsMenu();
188         }
189     }
190
191 }
```

FragCuentas.java 1/5

```java
package cl.theroot.passbank.fragmento;

import android.app.AlertDialog;
import android.app.Fragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.ContextMenu;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ExpandableListView;

import java.util.ArrayList;
import java.util.List;

import cl.theroot.passbank.ActividadPrincipal;
import cl.theroot.passbank.adaptador.AdapCategoriasCuentas;
import cl.theroot.passbank.R;
import cl.theroot.passbank.dominio.Categoria;
import cl.theroot.passbank.dominio.CategoriaCuenta;
import cl.theroot.passbank.dominio.CategoriaListaCuentas;
import cl.theroot.passbank.dominio.Contrasenna;
import cl.theroot.passbank.dominio.Cuenta;
import cl.theroot.passbank.dominio.CuentaConFecha;
import cl.theroot.passbank.datos.CategoriaCuentaDAO;
import cl.theroot.passbank.datos.CategoriaDAO;
import cl.theroot.passbank.datos.ContrasennaDAO;
import cl.theroot.passbank.datos.CuentaDAO;
import cl.theroot.passbank.datos.nombres.ColCuenta;

public class FragCuentas extends Fragment {
    private static final String TAG = "BdC-AdapCuentas";
    private AdapCategoriasCuentas adapter;

    private CuentaDAO cuentaDAO;
    private CategoriaDAO categoriaDAO;
    private CategoriaCuentaDAO categoriaCuentaDAO;
    private ContrasennaDAO contrasennaDAO;

    private List<CategoriaListaCuentas> listaCategorias = new
    ArrayList<>();

    public Cuenta selectedAccount;

    public FragCuentas() {

    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
```

FragCuentas.java 2/5

```
54          setHasOptionsMenu(true);
55          cuentaDAO = new CuentaDAO(getActivity().getApplicationContext
            ());
56          categoriaDAO = new CategoriaDAO(getActivity().
            getApplicationContext());
57          categoriaCuentaDAO = new CategoriaCuentaDAO(getActivity().
            getApplicationContext());
58          contrasennaDAO = new ContrasennaDAO(getActivity().
            getApplicationContext());
59
60          View view = inflater.inflate(R.layout.fragmento_cuentas, null
            );
61          ExpandableListView expListView = view.findViewById(R.id.
            expListView);
62
63          fillAccountsInfo();
64
65          adapter = new AdapCategoriasCuentas(getActivity(),
            listaCategorias);
66          expListView.setAdapter(adapter);
67          registerForContextMenu(expListView);
68
69          expListView.setOnGroupClickListener(new ExpandableListView.
            OnGroupClickListener() {
70              @Override
71              public boolean onGroupClick(ExpandableListView
                expandableListView, View view, int i, long l) {
72                  return false;
73              }
74          });
75
76          expListView.setOnGroupExpandListener(new ExpandableListView.
            OnGroupExpandListener() {
77              @Override
78              public void onGroupExpand(int i) {
79
80              }
81          });
82
83          expListView.setOnGroupCollapseListener(new ExpandableListView
            .OnGroupCollapseListener() {
84              @Override
85              public void onGroupCollapse(int i) {
86
87              }
88          });
89
90          expListView.setOnChildClickListener(new ExpandableListView.
            OnChildClickListener() {
91              @Override
92              public boolean onChildClick(ExpandableListView
                expandableListView, View view, int i, int il, long l) {
93                  Cuenta cuenta = (Cuenta) adapter.getChild(i, il);
94                  FragDetalleCuenta fragDetalleCuenta = new
                    FragDetalleCuenta();
```

```
95                  Bundle bundle = new Bundle();
96                  bundle.putString(ColCuenta.NOMBRE.toString(), cuenta.
                    getNombre());
97                  fragDetalleCuenta.setArguments(bundle);
98                  return ((ActividadPrincipal) getActivity()).
                    cambiarFragmento(fragDetalleCuenta);
99              }
100         });
101
102         return view;
103     }
104
105
106     //Creación del submenu principal del fragmento
107     @Override
108     public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
         {
109         inflater.inflate(R.menu.sub_menu_cuentas, menu);
110         super.onCreateOptionsMenu(menu, inflater);
111     }
112
113     //Creación de la funcionalidad del submenu del fragmento
114     public boolean onOptionsItemSelected(MenuItem item) {
115         if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
             {
116             return false;
117         }
118         switch (item.getItemId()) {
119             case R.id.sub_menu_accounts_add:
120                 return ((ActividadPrincipal) getActivity()).
                    cambiarFragmento(new FragAgregarEditarCuenta());
121
122             default:
123                 return super.onOptionsItemSelected(item);
124         }
125     }
126
127
128     //Creación del menu contextual para click prolongado en una
        Account
129     @Override
130     public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenu.ContextMenuInfo menuInfo) {
131         super.onCreateContextMenu(menu, v, menuInfo);
132
133         ExpandableListView.ExpandableListContextMenuInfo info = (
            ExpandableListView.ExpandableListContextMenuInfo) menuInfo;
134         int type = ExpandableListView.getPackedPositionType(info.
            packedPosition);
135
136         if (type == ExpandableListView.PACKED_POSITION_TYPE_CHILD) {
137             MenuInflater inflater = getActivity().getMenuInflater();
138             inflater.inflate(R.menu.cont_menu_cuentas, menu);
139         }
140     }
```

FragCuentas.java 4/5

```
141
142     //Creación de la funcionalidad del menu contextual
143     @Override
144     public boolean onContextItemSelected(MenuItem item) {
145         if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
             {
146             return false;
147         }
148         ExpandableListView.ExpandableListContextMenuInfo info = (
            ExpandableListView.ExpandableListContextMenuInfo) item.
            getMenuInfo();
149         Integer groupIndex = ExpandableListView.
            getPackedPositionGroup(info.packedPosition);
150         Integer childIndex = ExpandableListView.
            getPackedPositionChild(info.packedPosition);
151         selectedAccount = adapter.getChild(groupIndex, childIndex);
152
153         switch (item.getItemId()) {
154             case R.id.cont_menu_accounts_edit:
155                 FragAgregarEditarCuenta fragAgregarEditarCuenta = new
                     FragAgregarEditarCuenta();
156                 Bundle args = new Bundle();
157                 args.putString(ColCuenta.NOMBRE.toString(),
                    selectedAccount.getNombre());
158                 fragAgregarEditarCuenta.setArguments(args);
159                 ((ActividadPrincipal) getActivity()).cambiarFragmento
                    (fragAgregarEditarCuenta);
160                 return true;
161
162             case R.id.cont_menu_accounts_delete:
163                 AlertDialog alertDialog = new AlertDialog.Builder(
                    getActivity()).create();
164                 alertDialog.setTitle("Eliminación de Cuenta");
165                 alertDialog.setMessage("¿Está seguro que desea
                    eliminar la cuenta " + selectedAccount.getNombre() +
                    "?");
166                 alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE,
                    "NO", new DialogInterface.OnClickListener() {
167                     public void onClick(DialogInterface dialog, int
                        which) {
168                         dialog.dismiss();
169                     }
170                 });
171                 alertDialog.setButton(AlertDialog.BUTTON_POSITIVE,
                    "SÍ", new DialogInterface.OnClickListener() {
172                     public void onClick(DialogInterface dialog, int
                        which) {
173                         if (cuentaDAO.eliminarUna(selectedAccount.
                            getNombre()) > 0) {
174                             fillAccountsInfo();
175                             adapter.notifyDataSetChanged();
176                             ((ActividadPrincipal) getActivity()).
                                actualizarBundles(Cuenta.class,
                                selectedAccount.getNombre(), null);
177                         }
```

```
178                         dialog.dismiss();
179                     }
180                 });
181                 alertDialog.show();
182                 return true;
183
184             default:
185                 return super.onContextItemSelected(item);
186         }
187     }
188
189     //Llenado de los datos a mostrar
190     private void fillAccountsInfo() {
191         listaCategorias.clear();
192         for (Categoria categoria : categoriaDAO.seleccionarTodas()) {
193             List<CuentaConFecha> cuentas = new ArrayList<>();
194             for (CategoriaCuenta categoriaCuenta : categoriaCuentaDAO
                .seleccionarPorCategoria(categoria.getNombre())) {
195                 Cuenta cuenta = cuentaDAO.seleccionarUna(
                    categoriaCuenta.getNombreCuenta());
196                 if (cuenta != null) {
197                     Contrasenna contrasenna = contrasennaDAO.
                        seleccionarUltimaPorCuenta(cuenta.getNombre());
198                     String fecha = "2000/01/01";
199                     if (contrasenna != null) {
200                         fecha = contrasenna.getFecha();
201                     }
202                     cuentas.add(new CuentaConFecha(cuenta.getNombre
                        (), cuenta.getDescripcion(), cuenta.getValidez(),
                         fecha));
203                 } else {
204                     Log.e(TAG, "No se encontró la cuenta con el
                        nombre: " + categoriaCuenta.getNombreCuenta());
205                 }
206             }
207             listaCategorias.add(new CategoriaListaCuentas(categoria.
                getNombre(), categoria.getPosicion(), cuentas));
208         }
209
210         List<CuentaConFecha> cuentas = new ArrayList<>();
211         for (Cuenta cuenta : cuentaDAO.seleccionarTodas()) {
212             Contrasenna contrasenna = contrasennaDAO.
                seleccionarUltimaPorCuenta(cuenta.getNombre());
213             String fecha = "2000/01/01";
214             if (contrasenna != null) {
215                 fecha = contrasenna.getFecha();
216             }
217             cuentas.add(new CuentaConFecha(cuenta.getNombre(), cuenta
                .getDescripcion(), cuenta.getValidez(), fecha));
218         }
219         listaCategorias.add(new CategoriaListaCuentas("", null,
            cuentas));
220     }
221 }
```

FragDetalleCategoria.java 1/4

```java
package cl.theroot.passbank.fragmento;


import android.app.AlertDialog;
import android.app.Fragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

import cl.theroot.passbank.ActividadPrincipal;
import cl.theroot.passbank.adaptador.AdapCuentas;
import cl.theroot.passbank.R;
import cl.theroot.passbank.dominio.Categoria;
import cl.theroot.passbank.dominio.CategoriaCuenta;
import cl.theroot.passbank.dominio.Cuenta;
import cl.theroot.passbank.datos.CategoriaCuentaDAO;
import cl.theroot.passbank.datos.CategoriaDAO;
import cl.theroot.passbank.datos.CuentaDAO;
import cl.theroot.passbank.datos.nombres.ColCategoria;


/**
 * A simple {@link Fragment} subclass.
 */
public class FragDetalleCategoria extends Fragment {
    private AdapCuentas adaptador;
    private List<Cuenta> cuentas = new ArrayList<>();
    private Bundle bundle;
    private String nombreCategoria = null;

    private CategoriaDAO categoriaDAO;
    private CategoriaCuentaDAO categoriaCuentaDAO;
    private CuentaDAO cuentaDAO;

    public FragDetalleCategoria() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
        setHasOptionsMenu(true);
        cuentaDAO = new CuentaDAO(getActivity().getApplicationContext
        ());
```

FragDetalleCategoria.java 2/4

```
54          categoriaCuentaDAO = new CategoriaCuentaDAO(getActivity().
            getApplicationContext());
55          categoriaDAO = new CategoriaDAO(getActivity().
            getApplicationContext());
56
57          View view = inflater.inflate(R.layout.
            fragmento_detalle_categoria, null);
58          TextView TV_subTitulo = view.findViewById(R.id.TV_subTitulo);
59          TextView TV_name = view.findViewById(R.id.TV_name);
60          ListView listView = view.findViewById(R.id.LV_cuentas);
61
62          bundle = getArguments();
63          if (bundle != null) {
64              nombreCategoria = bundle.getString(ColCategoria.NOMBRE.
                toString());
65              if (nombreCategoria != null) {
66                  TV_name.setText(nombreCategoria);
67                  for (CategoriaCuenta categoriaCuenta :
                    categoriaCuentaDAO.seleccionarPorCategoria(
                    nombreCategoria)) {
68                      Cuenta cuenta = cuentaDAO.seleccionarUna(
                        categoriaCuenta.getNombreCuenta());
69                      if (cuenta != null) {
70                          cuentas.add(cuenta);
71                      }
72                  }
73              }
74          }
75
76          adaptador = new AdapCuentas(getActivity(), cuentas,
            nombreCategoria);
77          listView.setAdapter(adaptador);
78          listView.setSelector(android.R.color.transparent);
79
80          if (cuentas.isEmpty()) {
81              TV_subTitulo.setVisibility(View.INVISIBLE);
82          }
83          getActivity().invalidateOptionsMenu();
84          return view;
85      }
86
87      @Override
88      public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
         {
89          inflater.inflate(R.menu.sub_menu_detalle_categoria, menu);
90          super.onCreateOptionsMenu(menu, inflater);
91      }
92
93      @Override
94      public void onPrepareOptionsMenu(Menu menu){
95          if (adaptador != null) {
96              menu.findItem(R.id.
                sub_menu_detalle_categoria_habilitar_orden).setEnabled(
                adaptador.getCount() > 1);
97          }
```

FragDetalleCategoria.java 3/4

```
98          super.onPrepareOptionsMenu(menu);
99      }
100
101     //Creación de la funcionalidad del submenu del fragmento
102     public boolean onOptionsItemSelected(MenuItem item) {
103         if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
                {
104             return false;
105         }
106         switch (item.getItemId()) {
107             case R.id.sub_menu_detalle_categoria_volver:
108                 getActivity().onBackPressed();
109                 return true;
110             case R.id.sub_menu_detalle_categoria_editar:
111                 FragAgregarEditarCategoria fragAgregarEditarCategoria
                        = new FragAgregarEditarCategoria();
112                 Bundle args = new Bundle();
113                 args.putString(ColCategoria.NOMBRE.toString(),
                        nombreCategoria);
114                 fragAgregarEditarCategoria.setArguments(args);
115                 return ((ActividadPrincipal) getActivity()).
                        cambiarFragmento(fragAgregarEditarCategoria);
116             case R.id.sub_menu_detalle_categoria_eliminar:
117                 AlertDialog alertDialog = new AlertDialog.Builder(
                        getActivity()).create();
118                 alertDialog.setTitle("Eliminacón de una Categoría");
119                 alertDialog.setMessage("¿Está seguro que desea
                        eliminar la categoria " + nombreCategoria + "? Las
                        cuentas asociadas no se eliminarán, pero ya no se
                        relacionarán con esta categoría.");
120                 alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE,
                        "NO", new DialogInterface.OnClickListener() {
121                     public void onClick(DialogInterface dialog, int
                            which) {
122                         dialog.dismiss();
123                     }
124                 });
125                 alertDialog.setButton(AlertDialog.BUTTON_POSITIVE,
                        "SÍ", new DialogInterface.OnClickListener() {
126                     public void onClick(DialogInterface dialog, int
                            which) {
127                         if (categoriaDAO.eliminarUna(nombreCategoria)
                                > 0) {
128                             ((ActividadPrincipal) getActivity()).
                                cambiarFragmento(new FragCategorias());
129                             ((ActividadPrincipal) getActivity()).
                                actualizarBundles(Categoria.class,
                                nombreCategoria, null);
130                         }
131                         dialog.dismiss();
132                     }
133                 });
134                 alertDialog.show();
135                 return true;
136             case R.id.sub_menu_detalle_categoria_habilitar_orden:
```

FragDetalleCategoria.java 4/4

```
137                 item.setChecked(!item.isChecked());
138                 if (item.isChecked()) {
139                     adaptador.setOcultarFlechas(false);
140                 } else {
141                     adaptador.setOcultarFlechas(true);
142                 }
143                 return true;
144             default:
145                 return super.onOptionsItemSelected(item);
146         }
147     }
148
149 }
```

FragExportar.java 1/10

```java
package cl.theroot.passbank.fragmento;

import android.accounts.AccountManager;
import android.app.Fragment;
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.IntentSender;
import android.content.pm.ActivityInfo;
import android.content.res.Configuration;
import android.media.VolumeShaper;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.text.Editable;
import android.text.TextWatcher;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

import com.google.android.gms.auth.GoogleAuthUtil;
import com.google.android.gms.common.AccountPicker;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GoogleApiAvailability;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.ResultCallback;
import com.google.android.gms.common.api.Status;
import com.google.android.gms.drive.Drive;
import com.google.android.gms.drive.DriveApi;
import com.google.android.gms.drive.DriveContents;
import com.google.android.gms.drive.DriveFile;
import com.google.android.gms.drive.DriveFolder;
import com.google.android.gms.drive.DriveId;
import com.google.android.gms.drive.Metadata;
import com.google.android.gms.drive.MetadataBuffer;
import com.google.android.gms.drive.MetadataChangeSet;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;

import cl.theroot.passbank.ActividadPrincipal;
import cl.theroot.passbank.CustomToast;
import cl.theroot.passbank.R;
import cl.theroot.passbank.dominio.Parametro;
```

FragExportar.java 2/10

```
56  import cl.theroot.passbank.datos.ParametroDAO;
57  import cl.theroot.passbank.datos.nombres.NombreBD;
58  import cl.theroot.passbank.datos.nombres.NombreParametro;
59
60  import static android.app.Activity.RESULT_OK;
61
62  /**
63   * A simple {@link Fragment} subclass.
64   */
65  public class FragExportar extends Fragment implements View.
    OnClickListener, GoogleApiClient.ConnectionCallbacks, GoogleApiClient
    .OnConnectionFailedListener{
66      private static final String TAG = "BdC-FragExportar";
67      private static final int SELECCIONAR_CUENTA = 1;
68      private static final int SOLUCIONADOR_PROBLEMAS = 2;
69
70      private EditText ET_cuentaSeleccionada;
71      private TextView TV_instrucciones;
72      private ProgressDialog mensajeProgreso;
73
74      public GoogleApiClient mGoogleApiClient;
75
76      private ParametroDAO parametroDAO;
77
78      public FragExportar() {
79          // Required empty public constructor
80      }
81
82
83      @Override
84      public View onCreateView(LayoutInflater inflater, ViewGroup
        container, Bundle savedInstanceState) {
85          setHasOptionsMenu(true);
86          parametroDAO = new ParametroDAO(getActivity().
            getApplicationContext());
87
88          View view = inflater.inflate(R.layout.fragmento_exportar,
            null);
89          ET_cuentaSeleccionada = view.findViewById(R.id.
            ET_cuenta_seleccionada);
90          ET_cuentaSeleccionada.setOnClickListener(this);
91          ET_cuentaSeleccionada.addTextChangedListener(new TextWatcher
            () {
92              @Override
93              public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
94
95              }
96
97              @Override
98              public void onTextChanged(CharSequence s, int start, int
                before, int count) {
99
100             }
101
```

FragExportar.java 3/10

```
102             @Override
103             public void afterTextChanged(Editable s) {
104                 getActivity().invalidateOptionsMenu();
105             }
106         });
107         ImageView IV_vaciar_usuario = view.findViewById(R.id.
            IV_vaciar_usuario);
108         IV_vaciar_usuario.setOnClickListener(this);
109         TV_instrucciones = view.findViewById(R.id.TV_instruccion);
110         TV_instrucciones.setVisibility(View.INVISIBLE);
111         mensajeProgreso = new ProgressDialog(getActivity());
112         mensajeProgreso.setTitle("Creando Respaldo");
113         mensajeProgreso.setMessage("Su respaldo está siendo creado,
            favor esperar...");
114         mensajeProgreso.setCancelable(false);
115
116         Parametro parametro = parametroDAO.seleccionarUno(
            NombreParametro.CUENTA_GOOGLE.toString());
117         if (parametro != null && !parametro.getValor().equals("")) {
118             ET_cuentaSeleccionada.setText(parametro.getValor());
119             TV_instrucciones.setVisibility(View.VISIBLE);
120         }
121         getActivity().invalidateOptionsMenu();
122         return view;
123     }
124
125     @Override
126     public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
             {
127         inflater.inflate(R.menu.sub_menu_exportar, menu);
128         super.onCreateOptionsMenu(menu, inflater);
129     }
130
131     @Override
132     public void onPrepareOptionsMenu(Menu menu){
133         if (ET_cuentaSeleccionada != null) {
134             menu.findItem(R.id.sub_menu_exportar_respaldar).
                setEnabled(!ET_cuentaSeleccionada.getText().toString().
                isEmpty());
135         }
136         super.onPrepareOptionsMenu(menu);
137     }
138
139     @Override
140     public boolean onOptionsItemSelected(MenuItem item) {
141         if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
             {
142             return false;
143         }
144         switch (item.getItemId()) {
145             case R.id.sub_menu_exportar_back:
146                 getActivity().onBackPressed();
147                 return true;
148             case R.id.sub_menu_exportar_respaldar:
```

```
149                 String email = ET_cuentaSeleccionada.getText().
                    toString();
150                 if (email.isEmpty()) {
151                     new CustomToast(getActivity().
                        getApplicationContext(), "¡Acción
                        Cancelada!\nDebe seleccionar una cuenta antes de
                        realizar el respaldo.");
152                     return true;
153                 }
154
155                 if (mGoogleApiClient != null && mGoogleApiClient.
                    isConnected()) {
156                     new CustomToast(getActivity().
                        getApplicationContext(), "¡Acción Cancelada!\nYa
                        se está ejecutando la creación del respaldo de
                        los datos.");
157                     return true;
158                 }
159
160                 //Deshabilitar rotación de pantalla...
161                 if (getActivity().getResources().getConfiguration().
                    orientation == Configuration.ORIENTATION_LANDSCAPE) {
162                     getActivity().setRequestedOrientation(
                        ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
163                 } else {
164                     getActivity().setRequestedOrientation(
                        ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
165                 }
166
167                 parametroDAO.actualizarUna(new Parametro(
                    NombreParametro.CUENTA_GOOGLE.toString(), email, null
                    ));
168                 mensajeProgreso.show();
169                 mGoogleApiClient = new GoogleApiClient.Builder(
                    getActivity())
170                         .addApi(Drive.API)
171                         .addScope(Drive.SCOPE_APPFOLDER)
172                         .addConnectionCallbacks(this)
173                         .addOnConnectionFailedListener(this)
174                         .setAccountName(email)
175                         .build();
176                 mGoogleApiClient.connect();
177
178                 return true;
179             default:
180                 return super.onOptionsItemSelected(item);
181         }
182     }
183
184     @Override
185     public void onClick(View view) {
186         switch (view.getId()) {
187             case R.id.ET_cuenta_seleccionada:
```

```
188                 startActivityForResult(AccountPicker.
                    newChooseAccountIntent(null, null, new String[]{
                    GoogleAuthUtil.GOOGLE_ACCOUNT_TYPE}, true, null, null
                    , null, null), SELECCIONAR_CUENTA);
189                 return;
190
191             case R.id.IV_vaciar_usuario:
192                 ET_cuentaSeleccionada.setText("");
193                 TV_instrucciones.setVisibility(View.INVISIBLE);
194         }
195     }
196
197     @Override
198     public void onActivityResult(final int requestCode, final int
        resultCode, final Intent data) {
199         super.onActivityResult(requestCode, resultCode, data);
200         Log.i(TAG, "Resultados de Actividad - RequestCode: " +
            requestCode + " - ResultCode: " + requestCode);
201         if (requestCode == SELECCIONAR_CUENTA && resultCode ==
            RESULT_OK) {
202             String email = data.getStringExtra(AccountManager.
                KEY_ACCOUNT_NAME);
203             ET_cuentaSeleccionada.setText(email);
204             TV_instrucciones.setVisibility(View.VISIBLE);
205         }
206         if (requestCode == SOLUCIONADOR_PROBLEMAS) {
207             if (resultCode == RESULT_OK) {
208                 Log.i(TAG, "Problema solucionado... Reconectando...");
209                 mGoogleApiClient.connect();
210             } else {
211                 Log.e(TAG, "No se pudo solucionar el problema...
                    Desconectando...");
212                 terminarConexion("¡Error!\nNo se pudo realizar su
                    respaldo, intente más tarde.");
213             }
214         }
215     }
216
217     @Override
218     public void onConnected(@Nullable Bundle bundle) {
219         Log.i(TAG, "La conexión GoogleApiClient se realizó
            exitosamente");
220         //Obtenemos el archivo de la base de datos local
221         final java.io.File dbFile = getActivity().
            getApplicationContext().getDatabasePath(NombreBD.
            BANCO_CONTRASENNAS.toString());
222         if (dbFile != null) {
223             Log.i(TAG, "Directory: " + dbFile.toString());
224             // Se obtiene lo almacenado en Google Drive para
                posterior eliminación
225             final DriveFolder appFolder = Drive.DriveApi.getAppFolder
                (mGoogleApiClient);
```

FragExportar.java 6/10

```
226             appFolder.listChildren(mGoogleApiClient).
                setResultCallback(new ResultCallback<DriveApi.
                MetadataBufferResult>() {
227                 @Override
228                 public void onResult(@NonNull final DriveApi.
                    MetadataBufferResult metadataBufferResult) {
229                     if (!metadataBufferResult.getStatus().
                        isSuccess()) {
230                         Log.e(TAG, "¡Error! Hubo un problema al
                            tratar de recuperar los archivos desde
                            Google Drive.");
231                         terminarConexion("¡Error!\nNo se pudo
                            obtener los archivos de Google Drive,
                            intente más tarde.");
232                         return;
233                     }
234                     //Se genera una lista de IDs para su
                        posterior eliminación
235                     final List<DriveId> listaIds = new ArrayList
                        <>();
236                     Log.i(TAG,"¡Lista de Archivos! Archivos
                        obtenidos desde Google Drive:");
237                     MetadataBuffer metadataBuffer =
                        metadataBufferResult.getMetadataBuffer();
238                     for (Metadata metadata : metadataBuffer) {
239                         Log.i(TAG, metadata.getTitle() + " : " +
                            metadata.getDriveId());
240                         listaIds.add(metadata.getDriveId());
241                     }
242                     metadataBuffer.release();
243
244                     //Se comienza la creación del nuevo respaldo
245                     Drive.DriveApi.newDriveContents(
                        mGoogleApiClient).setResultCallback(new
                        ResultCallback<DriveApi.DriveContentsResult
                        >() {
246                         @Override
247                         public void onResult(@NonNull DriveApi.
                            DriveContentsResult driveContentsResult) {
248                             if (!driveContentsResult.getStatus().
                                isSuccess()) {
249                                 Log.e(TAG, "Error al obtener una
                                    instancia de contenidos");
250                                 terminarConexion("¡Error!\nNo se
                                    pudo crear un espacio para el
                                    respaldo en Google Drive,
                                    intente más tarde.");
251                                 return;
252                             }
253
254                             //Se crean los metadatos para el
                                nuevo respaldo
255                             final MetadataChangeSet
                                metadataChangeSet = new
                                MetadataChangeSet.Builder()
```

```
256                     .setTitle(NombreBD.
                        BANCO_CONTRASENNAS_RESPALDO.
                        toString())
257                     .setMimeType(
                        "application/x-sqlite3")
258                     .build();
259
260                 //Se copian los datos de la base de
                    datos local en el contenido para el
                    nuevo respaldo
261                 DriveContents contents =
                    driveContentsResult.getDriveContents
                    ();
262                 InputStream in = null;
263                 OutputStream out = null;
264                 try {
265                     out = contents.getOutputStream();
266                     in = new FileInputStream(dbFile);
267
268                     byte[] buffer = new byte[1024];
269                     int read;
270                     while ((read = in.read(buffer))
                        != -1) {
271                         out.write(buffer, 0, read);
272                     }
273                 } catch(FileNotFoundException ex) {
274                     Log.e(TAG, "No se encontró el
                        archivo local de la base de
                        datos", ex);
275                     terminarConexion("¡Error!\nNo se
                        pudo realizar su respaldo,
                        intente más tarde.");
276                     return;
277                 } catch(IOException ex) {
278                     Log.e(TAG, "No se pudo hacer
                        lectura de la base de datos", ex);
279                     terminarConexion("¡Error!\nNo se
                        pudo realizar su respaldo,
                        intente más tarde.");
280                     return;
281                 } finally {
282                     if (in != null) {
283                         try {
284                             in.close();
285                         } catch (IOException ex) {
286                             //Nada
287                         }
288                     }
289                     if (out != null) {
290                         try {
291                             out.close();
292                         } catch (IOException ex) {
293                             //Nada
294                         }
295                     }
```

```
296                 }
297
298                 //Se crea el nuevo respaldo, con los
                    metadatos y contenidos obtenidos
                    anteriormente
299                 appFolder.createFile(mGoogleApiClient
                    , metadataChangeSet, contents).
                    setResultCallback(new ResultCallback<
                    DriveFolder.DriveFileResult>() {
300                     @Override
301                     public void onResult(@NonNull
                        DriveFolder.DriveFileResult
                        driveFileResult) {
302                         if (!driveFileResult.
                            getStatus().isSuccess()) {
303                             Log.e(TAG, "Error al
                                crear el archivo de
                                respaldo en Google Drive"
                                );
304                             terminarConexion(
                                "¡Error!\nNo se pudo
                                crear el archivo de
                                respaldo en Google
                                Drive, intente más
                                tarde.");
305                             return;
306                         }
307                         Log.i(TAG, "El archivo de
                            respaldo fue creado
                            exitosamente con ID: " +
                            driveFileResult.getDriveFile
                            ().getDriveId().toString());
308
309                         //Se comienza la eliminación
                            de los respaldos anteriores
                            al recien creado
310                         if (!listaIds.isEmpty()) {
311                             final List<DriveId>
                                idsProcesados = new
                                ArrayList<>();
312                             for (final DriveId
                                driveId : listaIds) {
313                                 Log.i(TAG,
                                    "Eliminando el
                                    archivo con ID: " +
                                    driveId.toString());
314                                 DriveFile driveFile =
                                     driveId.asDriveFile
                                    ();
315                                 driveFile.delete(
                                    mGoogleApiClient).
                                    setResultCallback(new
                                     ResultCallback<
                                    Status>() {
316                                     @Override
```

```
317                                                     public void onResult(
                                                        @NonNull Status
                                                        status) {
318                                                         idsProcesados.add
                                                            (driveId);
319                                                         if (!status.
                                                            isSuccess()) {
320                                                             Log.e(TAG,
    "Error al eliminar el archivo con ID: " + driveId.toString());
321                                                         } else {
322                                                             Log.i(TAG,
    "Se eliminó el archivo con ID: " + driveId.toString());
323                                                         }
324                                                         if (listaIds.size
                                                            () ==
                                                            idsProcesados.
                                                            size()) {
325                                                             Log.i(TAG,
    "¡Éxito al Respaldar! Su respaldo fue creado exitosamente en Google
    Drive.");
326
    terminarConexion("¡Éxito al Respaldar!\nSu respaldo fue creado
    exitosamente en Google Drive.");
327                                                         }
328                                                     }
329                                                 });
330                                             }
331                                         } else {
332                                             Log.i(TAG, "¡Éxito al
                                                Respaldar! Su respaldo fue
                                                creado exitosamente en
                                                Google Drive.");
333                                             terminarConexion("¡Éxito al
                                                Respaldar!\nSu respaldo fue
                                                creado exitosamente en
                                                Google Drive.");
334                                         }
335
336                                     }
337                                 });
338                             }
339                         });
340                     }
341                 });
342             }
343         }
344
345         @Override
346         public void onConnectionSuspended(int i) {
347             Log.i(TAG, "La conexión GoogleApiClient se ha suspedido");
348         }
349
350         @Override
351         public void onConnectionFailed(@NonNull ConnectionResult
            connectionResult) {
```

```
352             Log.e(TAG, "La conexión GoogleApiClient ha fallado: " +
                connectionResult.toString());
353             if (!connectionResult.hasResolution()) {
354                 Log.e(TAG, "No existe una solución al problema... Cancelando
                    exportación...");
355                 terminarConexion(null);
356                 GoogleApiAvailability.getInstance().getErrorDialog(
                    getActivity(), connectionResult.getErrorCode(), 0).show();
357                 return;
358             }
359             try {
360                 Log.i(TAG, "Tratando de resolver el problema...");
361                 connectionResult.startResolutionForResult(getActivity(),
                    SOLUCIONADOR_PROBLEMAS);
362             } catch (IntentSender.SendIntentException ex) {
363                 Log.e(TAG, "Excepción mientras se trataba de iniciar la
                    actividad de resolución de problemas", ex);
364             }
365         }
366
367         private void terminarConexion(String mensaje) {
368             mGoogleApiClient.disconnect();
369             mGoogleApiClient = null;
370             mensajeProgreso.dismiss();
371             if (mensaje != null) {
372                 new CustomToast(getActivity().getApplicationContext(),
                    mensaje);
373             }
374             //Habilitar rotación de pantalla...
375             getActivity().setRequestedOrientation(ActivityInfo.
                SCREEN_ORIENTATION_UNSPECIFIED);
376         }
377     }
```

FragInicioSesion.java 1/2

```java
package cl.theroot.passbank.fragmento;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;

import cl.theroot.passbank.ActividadPrincipal;
import cl.theroot.passbank.Cifrador;
import cl.theroot.passbank.Desfragmentador;
import cl.theroot.passbank.ExcepcionBancoContrasennas;
import cl.theroot.passbank.R;
import cl.theroot.passbank.dominio.Parametro;
import cl.theroot.passbank.datos.CategoriaCuentaDAO;
import cl.theroot.passbank.datos.CategoriaDAO;
import cl.theroot.passbank.datos.ContrasennaDAO;
import cl.theroot.passbank.datos.CuentaDAO;
import cl.theroot.passbank.datos.PalabraDAO;
import cl.theroot.passbank.datos.ParametroDAO;
import cl.theroot.passbank.datos.nombres.NombreParametro;

public class FragInicioSesion extends Fragment implements  View.
OnClickListener{
    private static final String TAG = "BdC-FragInicioSesion";

    private EditText ET_contrasenna;

    private ParametroDAO parametroDAO;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container
    , Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragmento_inicio_sesion,
        null);
        Button B_login = view.findViewById(R.id.B_login);
        B_login.setOnClickListener(this);
        ET_contrasenna = view.findViewById(R.id.ET_password);
        getActivity().invalidateOptionsMenu();

        parametroDAO = new ParametroDAO(getActivity().
        getApplicationContext());

        //Tratar de crear la base de datos para el diccionario de palabras
        new PalabraDAO(getActivity().getApplicationContext());
        //Tratar de desfragmentar la base de datos de la aplicación
        Desfragmentador.intentarDesfragmentación(getActivity().
        getApplicationContext());

        //Imprimir datos de la base de datos para debuggeo
```

```
50          new CategoriaDAO(getActivity().getApplicationContext()).imprimir
            ();
51          new CuentaDAO(getActivity().getApplicationContext()).imprimir();
52          new CategoriaCuentaDAO(getActivity().getApplicationContext()).
            imprimir();
53          new ContrasennaDAO(getActivity().getApplicationContext()).
            imprimir();
54          new ParametroDAO(getActivity().getApplicationContext()).imprimir
            ();
55
56          return view;
57      }
58
59      @Override
60      public void onClick(View v) {
61          try {
62              switch (v.getId()) {
63                  case R.id.B_login:
64                      String contrasenna = ET_contrasenna.getText().
                        toString();
65                      if (contrasenna.isEmpty()) {
66                          throw new ExcepcionBancoContrasennas("Error -
                            Llave Maestra Requerida", "Se requiere el
                            ingreso de su Llave Maestra para el inicio de
                            sesión.");
67                      }
68
69                      Parametro parSalt = parametroDAO.seleccionarUno(
                        NombreParametro.SAL_HASH.toString());
70                      Parametro parHash = parametroDAO.seleccionarUno(
                        NombreParametro.RESULTADO_HASH.toString());
71
72                      String[] saltYHashObt = Cifrador.genHashedPass(
                        contrasenna, parSalt.getValor());
73                      if (!saltYHashObt[1].equals(parHash.getValor())) {
74                          throw new ExcepcionBancoContrasennas("Error -
                            Llave Maestra Incorrecta", "La Llave Maestra
                            ingresada no es correcta, favor intentar de
                            nuevo.");
75                      }
76
77                      Parametro parSaltEncr = parametroDAO.seleccionarUno(
                        NombreParametro.SAL_ENCRIPTACION.toString());
78                      String[] saltYHashEncr = Cifrador.genHashedPass(
                        contrasenna, parSaltEncr.getValor());
79                      ((ActividadPrincipal) getActivity()).userLogIn(
                        saltYHashEncr[1]);
80                      break;
81              }
82          } catch (ExcepcionBancoContrasennas ex) {
83              AlertDialog alertDialog = new AlertDialog.Builder(getActivity
                ()).create();
84              alertDialog.setTitle(ex.getTitulo());
85              alertDialog.setMessage(ex.getMensaje());
86              alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK", new
                DialogInterface.OnClickListener() {
87                  public void onClick(DialogInterface dialog, int which) {
88                      dialog.dismiss();
89                  }
90              });
91              alertDialog.show();
92          }
93      }
94  }
```