

2018

BANCO DE CONTRASEÑAS PARA SISTEMA OPERATIVO ANDROID VERSIÓN 4.1

DÍAZ EYSSAUTIER, BENJAMÍN CRISTÓBAL

<https://hdl.handle.net/11673/46183>

Repositorio Digital USM, UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

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

INTRODUCCIÓN	11
CAPÍTULO 1: ASPECTOS RELEVANTES DEL DISEÑO LÓGICO	13
1.1. DESCRIPCIÓN DE LA SITUACIÓN ACTUAL	15
1.2. PROBLEMAS DETECTADOS	26
1.3. DESCRIPCIÓN DEL SISTEMA PROPUESTO	28
1.3.1. OBJETIVOS Y BENEFICIOS DEL SISTEMA PROPUESTO	28
1.3.1.1. Objetivo Principal	28
1.3.1.2. Objetivos Específicos	28
1.3.1.3. Beneficios	28
1.3.2. DESCRIPCIÓN GENERAL DE LA SOLUCIÓN PROPUESTA	29
1.3.3. DIAGRAMA DE FLUJO ADMINISTRATIVO	30
1.3.4. ESTRUCTURA FUNCIONAL DEL SISTEMA	31
1.3.5. INFORMACIÓN A MANEJAR	33
1.3.5.1. Salidas del Sistema	33
1.3.5.2. Entradas del Sistema	34
1.3.5.3. Entidades de Información	35
1.3.6. ESTRUCTURA DE CÓDIGOS	36
1.3.7. CONDICIONANTES DE DISEÑO	36
CAPÍTULO 2: MEDIO AMBIENTE COMPUTACIONAL Y DESCRIPCIÓN DE ARCHIVOS	37
2.1. CARACTERÍSTICAS DEL RECURSO COMPUTACIONAL	39
2.1.1. CONFIGURACIÓN DEL SISTEMA	39
2.1.1.1. Tipo de Procesador	39
2.1.1.2. Tipo y Capacidad de Almacenamiento, y Mecanismos de Respaldo	39
2.1.1.3. Tipo y Número de Pantallas e Impresoras	39
2.1.2. SOFTWARE UTILIZADO	40
2.1.2.1. Sistema Operativo	40
2.1.2.2. Herramientas de Desarrollo de Software	41
2.2. DESCRIPCIÓN DE ARCHIVOS	42
2.2.1. DESCRIPCIÓN GENERAL DE ARCHIVOS	42
2.2.1.1. Categoría	42
2.2.1.2. Cuenta	42
2.2.1.3. Categoría/Cuenta	42
2.2.1.4. Contraseña	42

2.2.1.5.	Parámetro	42
2.2.2.	CODIFICACIÓN DE ARCHIVOS	43
2.2.3.	DESCRIPCIÓN ESPECÍFICA DE ARCHIVOS	43
2.2.3.1.	Categoría	43
2.2.3.2.	Cuenta	44
2.2.3.3.	Categoría/Cuenta	45
2.2.3.4.	Contraseña	45
2.2.3.5.	Parámetro	46
2.3.	DIAGRAMAS UML	47
2.3.1.	DESCRIPCIÓN DE CASOS DE USO	47
2.3.1.1.	Caso de Uso: Crear Cuenta	47
2.3.1.2.	Caso de Uso: Cambiar Llave Maestra	48
2.3.1.3.	Caso de Uso: Eliminar Categoría	48
2.3.2.	DIAGRAMAS DE CASOS DE USO	49
2.3.2.1.	Diagrama Caso de Uso: Crear Cuenta	49
2.3.2.2.	Diagrama Caso de Uso: Cambiar Llave Maestra	50
2.3.2.3.	Diagrama Caso de Uso: Eliminar Categoría	50
2.3.3.	DIAGRAMAS DE SECUENCIA	51
2.3.3.1.	Diagrama Secuencia: Crear Cuenta	51
2.3.3.2.	Diagrama Secuencia: Cambiar Llave Maestra	52
2.3.3.3.	Diagrama Secuencia: Eliminar Categoría	52
CAPÍTULO 3: ESTRUCTURA GENERAL DEL SISTEMA		53
3.1.	DIAGRAMA MODULAR	55
3.2.	DIAGRAMA DE MENÚS	56
3.3.	PROGRAMAS	59
3.3.1	LISTA DE PROGRAMAS	59
3.3.2.	DESCRIPCIÓN DE PROGRAMAS	62
3.3.2.1.	ActividadPrincipal.java	62
3.3.2.2.	FragAgregarEditarCuenta.java	64
3.3.2.3.	FragCambioLlaveMaestra.java	65
3.3.2.4.	FragConfiguracion.java	67
3.3.2.5.	FragCuentas.java	68
3.3.2.6.	FragDetalleCategoría.java	69
3.3.2.7.	FragExportar.java	71
3.3.2.8.	FragInicioSesion.java	72

CONCLUSIONES Y RECOMENDACIONES	75
BIBLIOGRAFÍA	77
ANEXOS	79

ÍNDICE DE GRÁFICOS

Gráfico 1-1.	Resultado pregunta N°1 de encuesta.	21
Gráfico 1-2.	Resultado pregunta N°2 de encuesta.	21
Gráfico 1-3.	Resultado pregunta N°3 de encuesta.	22
Gráfico 1-4.	Resultado pregunta N°4 de encuesta.	22
Gráfico 1-5.	Resultado pregunta N°5 de encuesta.	22
Gráfico 1-6.	Resultado pregunta N°6 de encuesta.	23
Gráfico 1-7.	Resultado pregunta N°7 de encuesta.	23
Gráfico 1-8.	Resultado pregunta N°8 de encuesta.	23
Gráfico 1-9.	Resultado pregunta N°9 de encuesta.	24
Gráfico 1-10.	Resultado pregunta N°10 de encuesta.	24
Gráfico 1-11.	Resultado pregunta N°11 de encuesta.	24

ÍNDICE DE FIGURAS

Figura 1-1.	Diagrama de Flujo Administrativo.	30
Figura 1-2.	Modelo Lógico del Sistema.	35
Figura 1-3.	Modelo Lógico de la Base de Datos “diccionario”.	36
Figura 2-1.	Diagrama de caso de uso de Crear_Cuenta.	49
Figura 2-2.	Diagrama de caso de uso de Cambiar_Llave_Maestra.	50
Figura 2-3.	Diagrama de caso de uso de Eliminar_Categoría.	50
Figura 2-4.	Diagrama de secuencia de Crear_Cuenta.	51
Figura 2-5.	Diagrama de secuencia de Cambiar_Llave_Maestra.	52
Figura 2-6.	Diagrama de secuencia de Eliminar_Categoría.	52
Figura 3-1.	Diagrama modular del sistema.	55
Figura 3-2.	Diseño del Menú de Aplicación.	56
Figura 3-3.	Diseño del Submenú de Aplicación, pantalla de Consultar Cuenta.	57
Figura 3-4.	Diseño del Menú Contextual, pantalla Cuentas.	57

Figura 3-5.	Diagrama de menús del sistema.	58
Figura 3-6.	Diagrama de bloques – Actividad Principal.	63
Figura 3-7.	Diseño de la Actividad Principal, con el Menú de Aplicación.	63
Figura 3-8.	Diagrama de bloques – Agregar/Editar Cuenta.	64
Figura 3-9.	Diseño de la pantalla Agregar/Editar Cuenta.	65
Figura 3-10.	Diagrama de bloques – Cambiar Llave Maestra.	65
Figura 3-11.	Diseño de la pantalla Cambiar Llave Maestra.	66
Figura 3-12.	Diagrama de bloques – Configuración.	67
Figura 3-13.	Diseño de la pantalla Configuración.	68
Figura 3-14.	Diagrama de bloques – Lista de Cuentas.	68
Figura 3-15.	Diseño de la pantalla Lista de Cuentas.	69
Figura 3-16.	Diagrama de bloques – Detalle de Categoría.	70
Figura 3-17.	Diseño de la pantalla Detalle de Categoría.	70
Figura 3-18.	Diagrama de bloques – Respalda Datos.	71
Figura 3-19.	Diseño de la pantalla Respalda Datos.	72
Figura 3-20.	Diagrama de bloques – Inicio de Sesión.	72
Figura 3-21.	Diseño de la pantalla Inicio de Sesión.	73

ÍNDICE DE TABLAS

Tabla 2-1.	Cantidad relativa de dispositivos que usan Android, según versión	40
Tabla 2-2.	Cuota de Mercado de Sistema Operativos para Smartphones, 4Q16.	41
Tabla 2-3.	Descripción de la tabla Categoría.	43
Tabla 2-4.	Descripción de la tabla Cuenta.	44
Tabla 2-5.	Descripción de la tabla Categoría/Cuenta.	45
Tabla 2-6.	Descripción de la tabla Contraseña.	45
Tabla 2-7.	Descripción de la tabla Parámetro.	46
Tabla 2-8.	Descripción del caso de uso Crear_Cuenta.	47
Tabla 2-9.	Descripción del caso de uso Cambiar_Llave_Maestra.	48
Tabla 2-10.	Descripción del caso de uso Eliminar_Categoría.	48

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 autenticación, 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.

LastPass:

- 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 N°1 de encuesta.

- ¿Para qué uso el celular?



Gráfico 1-2. Resultado pregunta N°2 de encuesta.

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

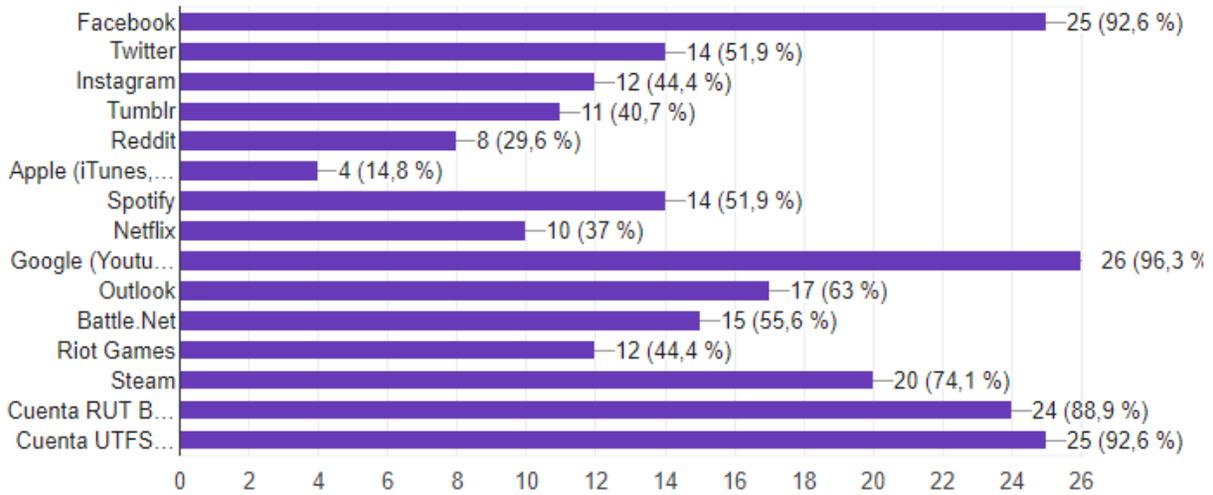


Gráfico 1-3. Resultado pregunta N°3 de encuesta.

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

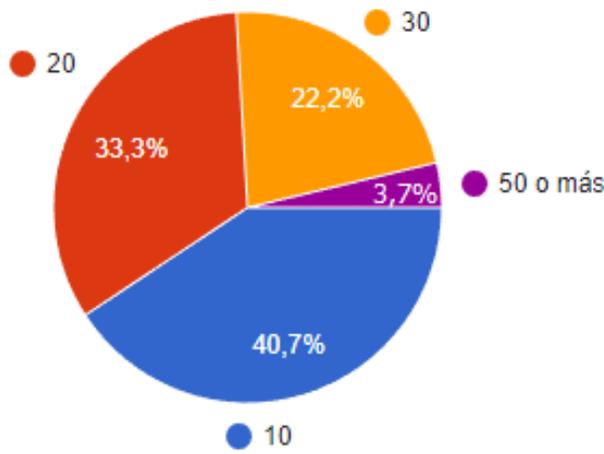


Gráfico 1-4. Resultado pregunta N°4 de encuesta.

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



Gráfico 1-5. Resultado pregunta N°5 de encuesta.

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

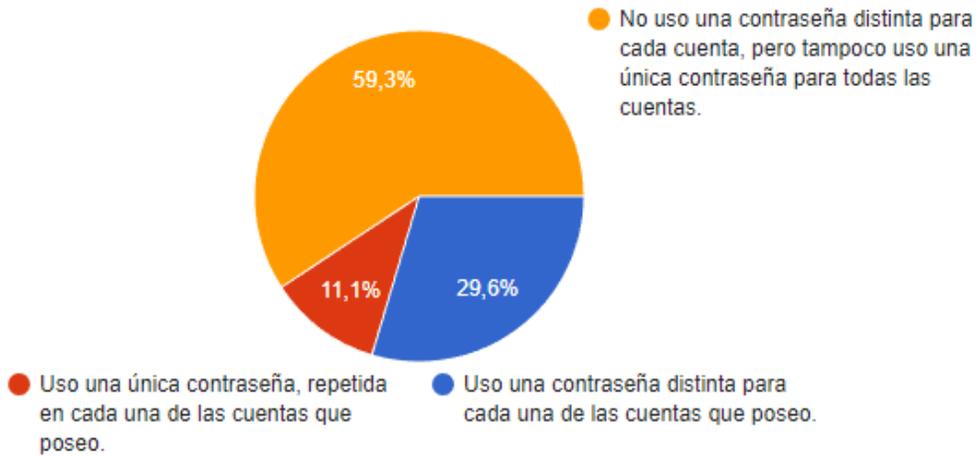


Gráfico 1-6. Resultado pregunta N°6 de encuesta.

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

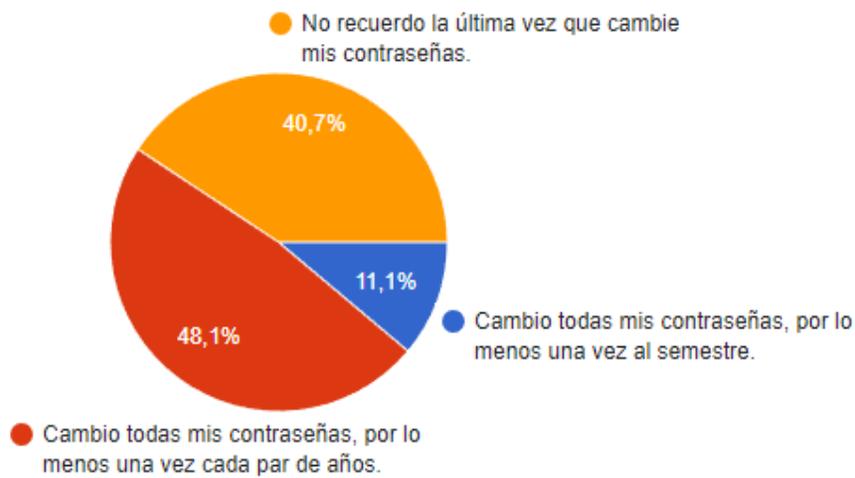


Gráfico 1-7. Resultado pregunta N°7 de encuesta.

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

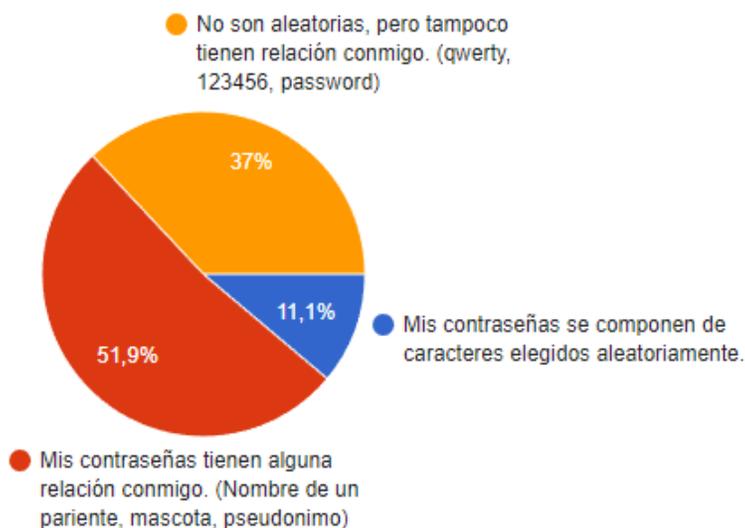


Gráfico 1-8. Resultado pregunta N°8 de encuesta.

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



Gráfico 1-9. Resultado pregunta N°9 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?

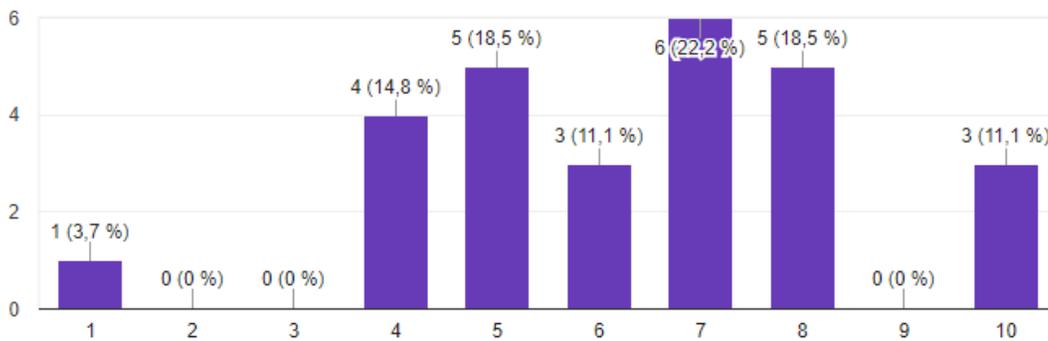


Gráfico 1-10. Resultado pregunta N°10 de encuesta.

- 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?

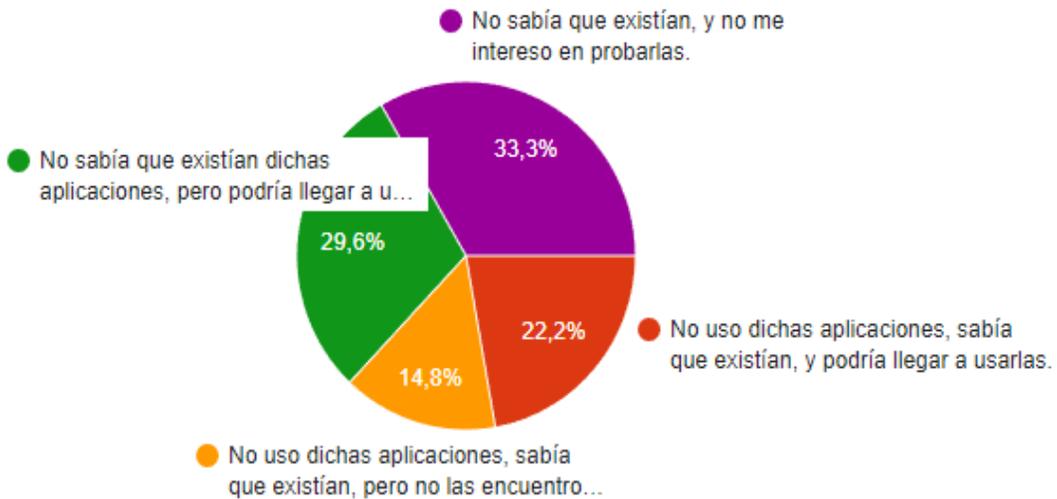


Gráfico 1-11. Resultado pregunta N°11 de encuesta.

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.
- Respalidar 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 descryptación 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

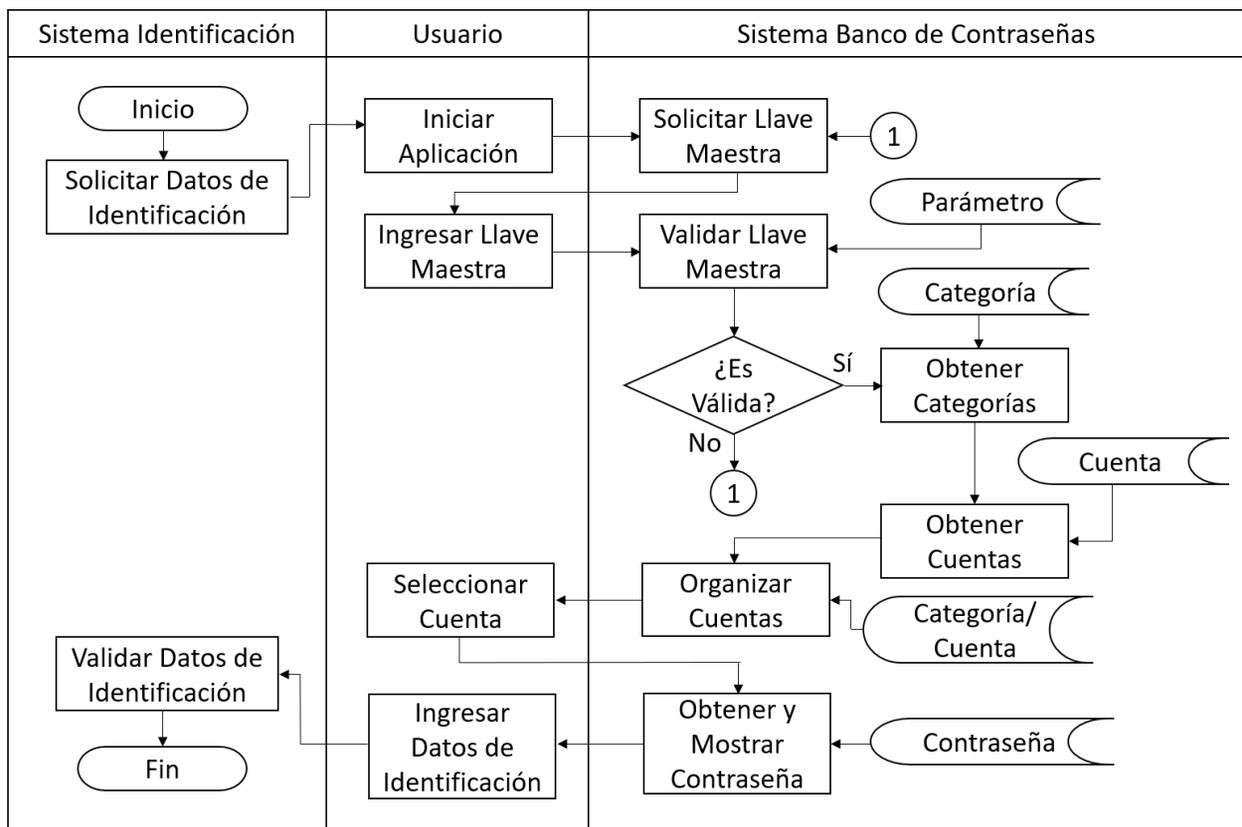


Figura 1-1. Diagrama de Flujo Administrativo.

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 autenticación como, por ejemplo, el sistema de autenticación utilizado por servicios de correo electrónico, con una dirección de correo y una contraseña, o el sistema de autenticación 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 descryptar 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 unir las.
- 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.

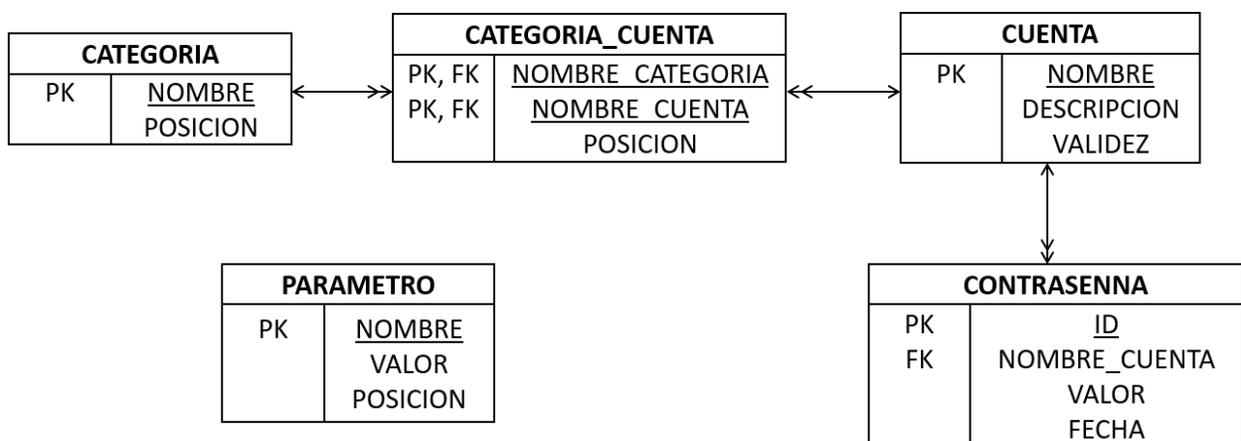


Figura 1-2. Modelo Lógico del Sistema.

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.

PALABRA	
PK	<u>ID</u> VALOR

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 autenticación 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 autenticarse, 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 considere 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 autenticado.
Escenario de éxito	
<p>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.</p>	

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 autenticado.
Escenario de éxito	
<p>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 autenticación de usuarios, con nuevos valores acordes a la nueva llave maestra. Se informa al cliente que su nueva llave maestra fue registrada exitosamente.</p>	
Escenario alternativo	
<p>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.</p> <p>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.</p>	
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 autenticado.
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

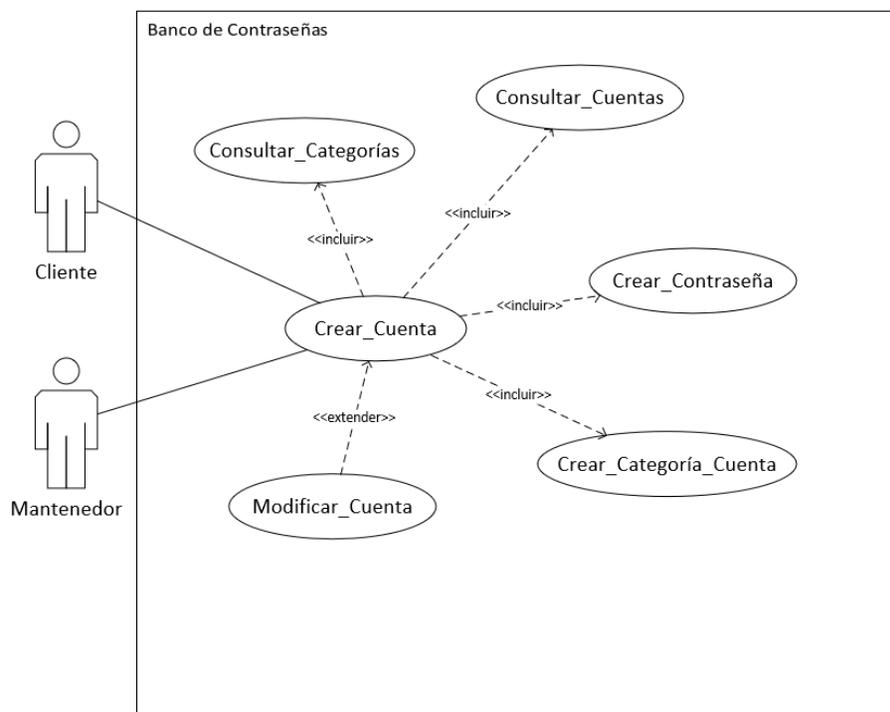


Figura 2-1. Diagrama de caso de uso de Crear_Cuenta.

2.3.2.2. Diagrama Caso de Uso: Cambiar Llave Maestra

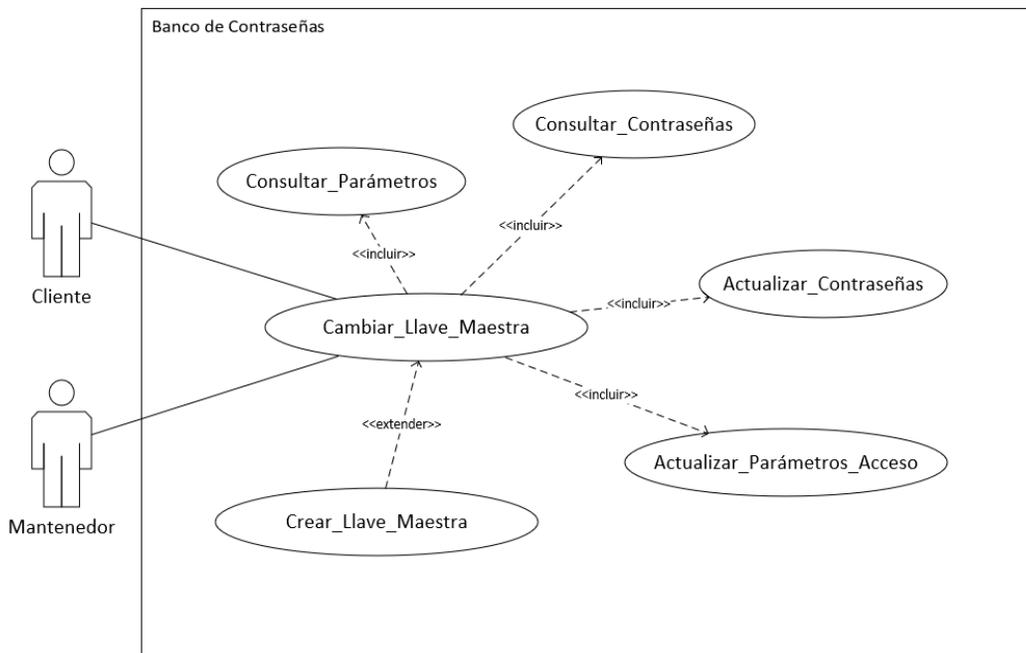


Figura 2-2. Diagrama de caso de uso de Cambiar_Llave_Maestra.

2.3.2.3. Diagrama Caso de Uso: Eliminar Categoría

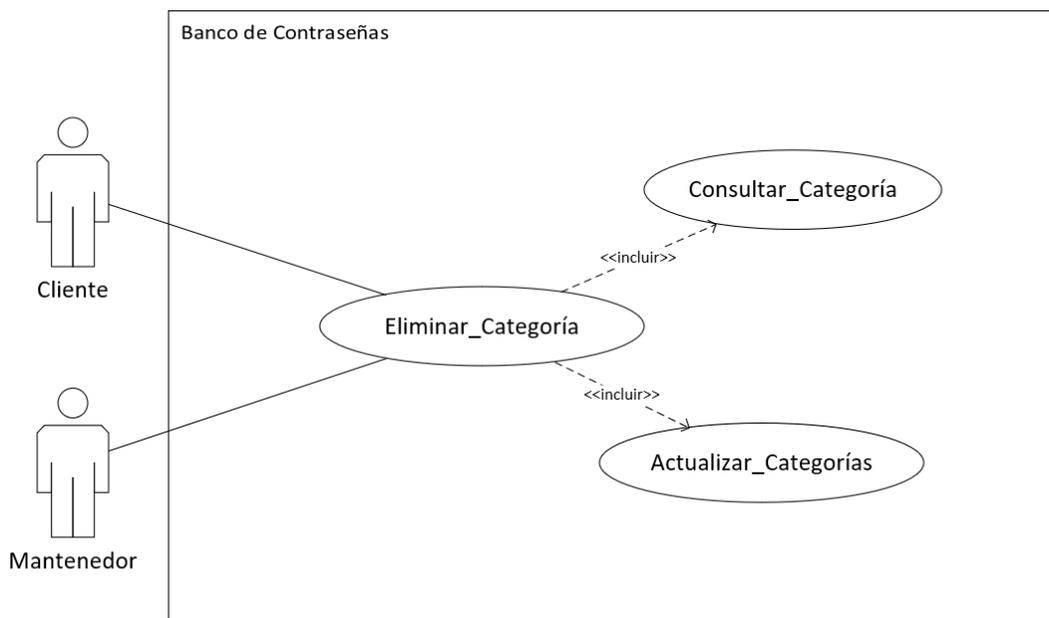


Figura 2-3. Diagrama de caso de uso de Eliminar_Categoría.

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 instancia 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

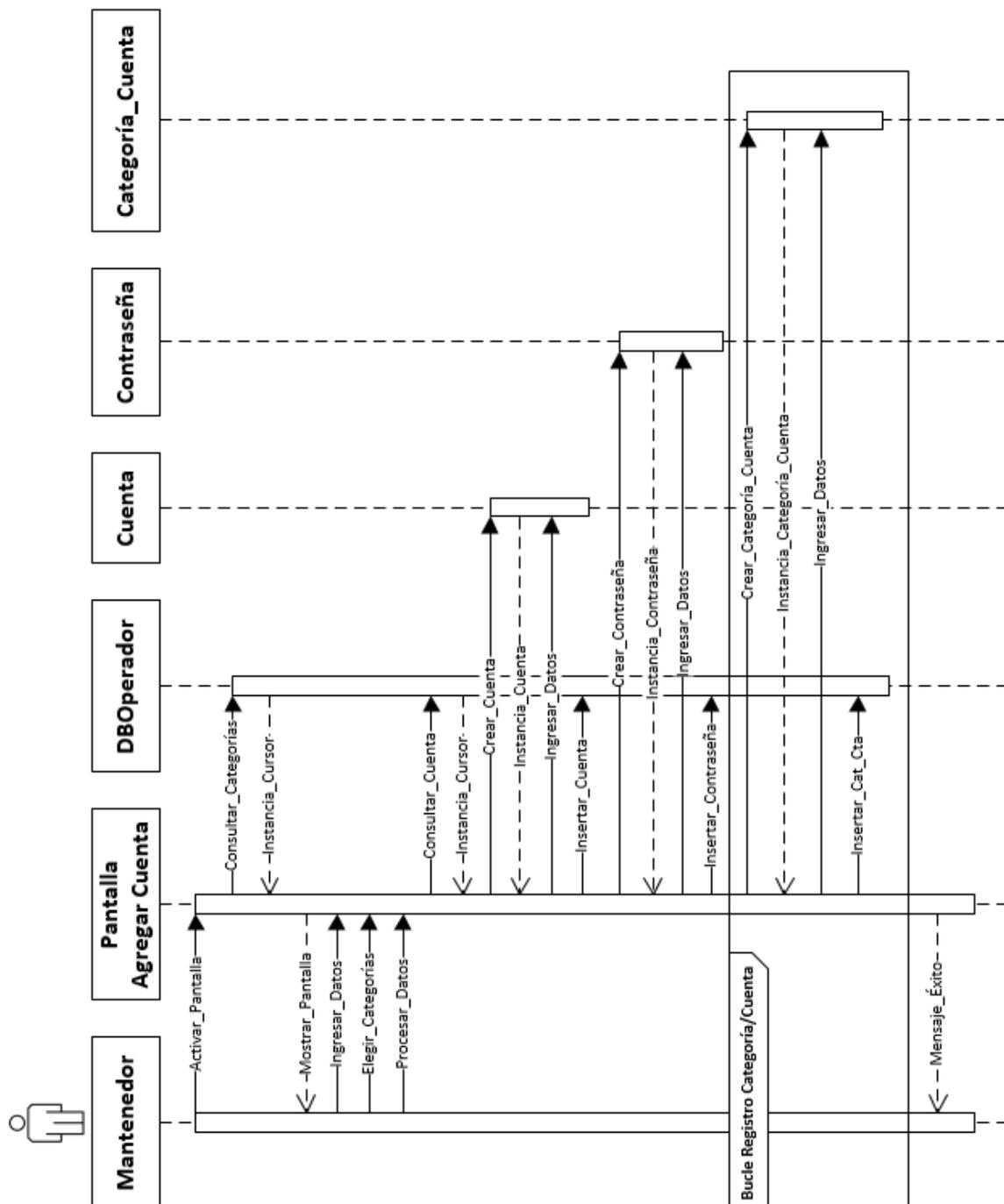


Figura 2-4. Diagrama de secuencia de Crear_Cuenta.

2.3.3.2. Diagrama Secuencia: Cambiar Llave Maestra

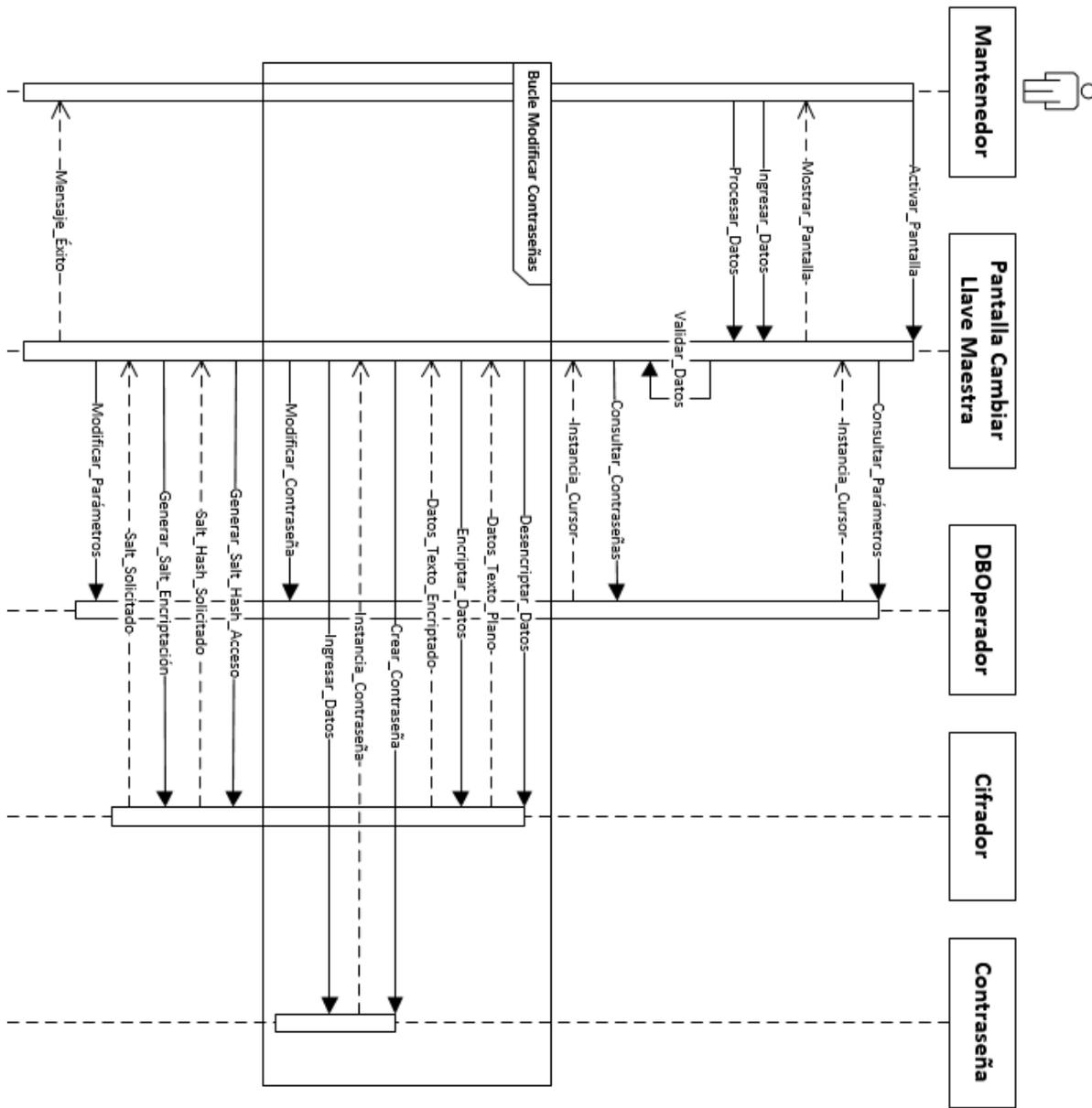


Figura 2-5. Diagrama de secuencia de Cambiar_Llave_Maestra.

2.3.3.3. Diagrama Secuencia: Eliminar Categoría

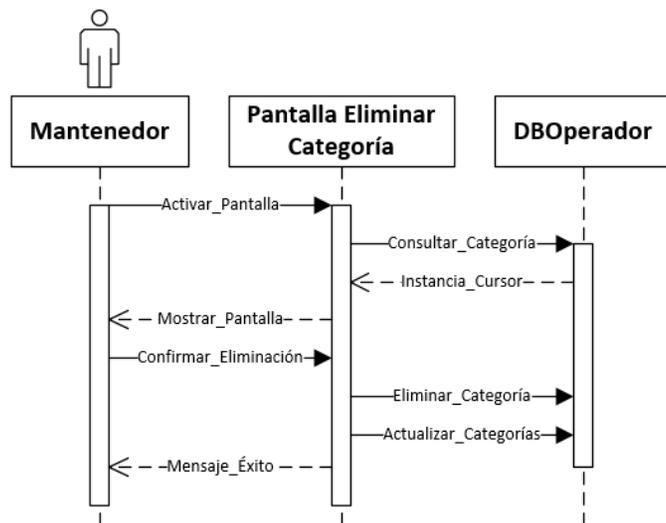


Figura 2-6. Diagrama de secuencia de Eliminar_Categoría.

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:

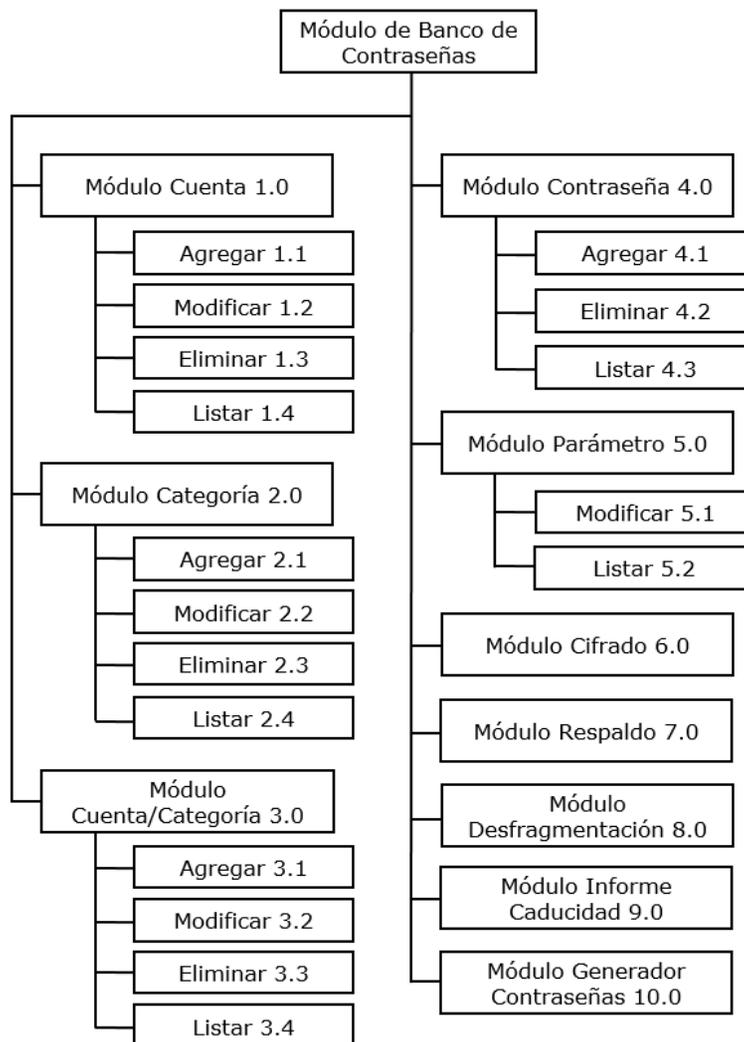


Figura 3-1. Diagrama modular del sistema.

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:

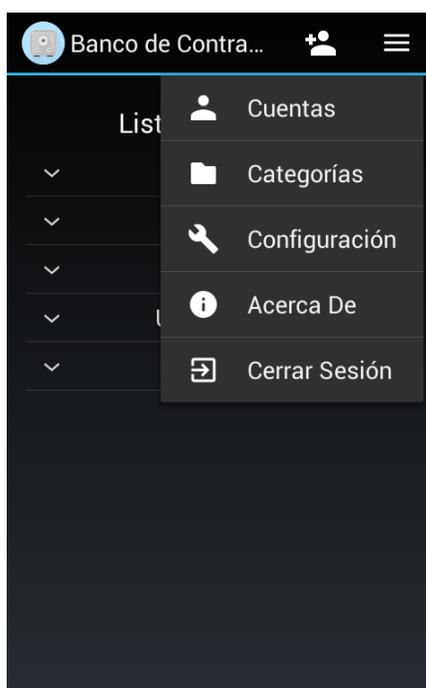


Figura 3-2. Diseño del Menú de Aplicación.

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:

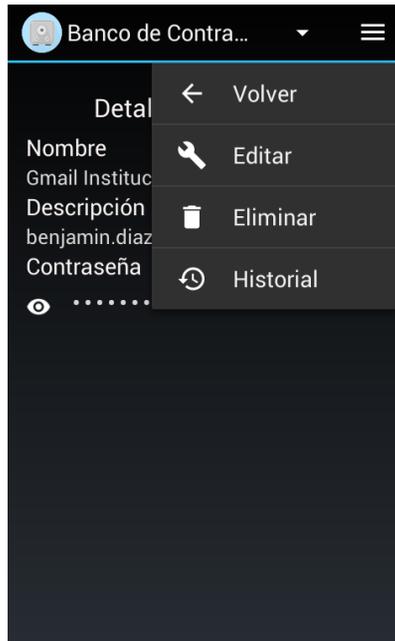


Figura 3-3. Diseño del Submenú de Aplicación, pantalla de Consultar Cuenta.

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:

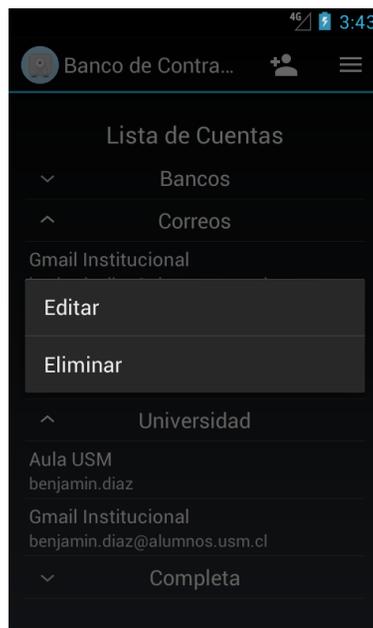


Figura 3-4. Diseño del Menú Contextual, pantalla Cuentas.

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:

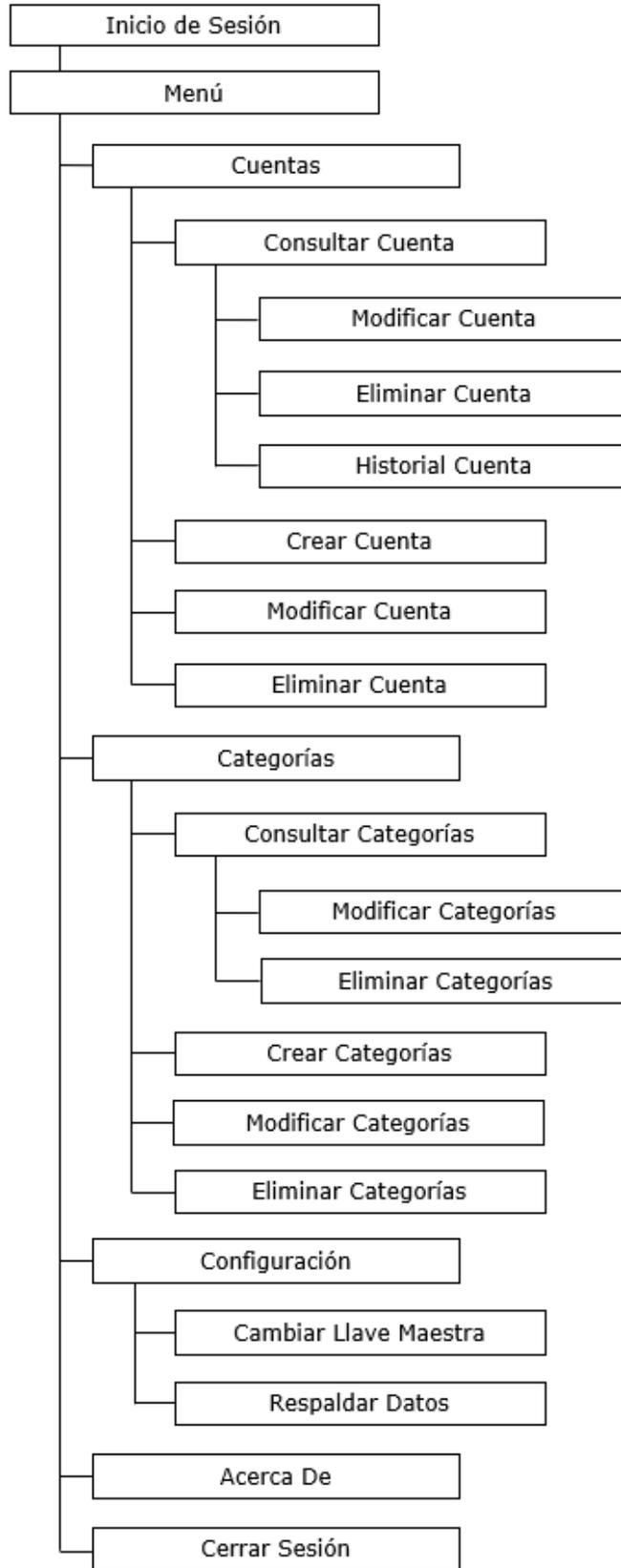


Figura 3-5. Diagrama de menús del sistema.

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.

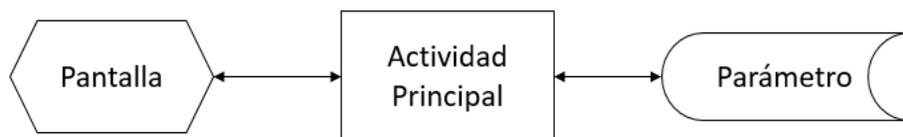
Diagrama de Bloques:

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.

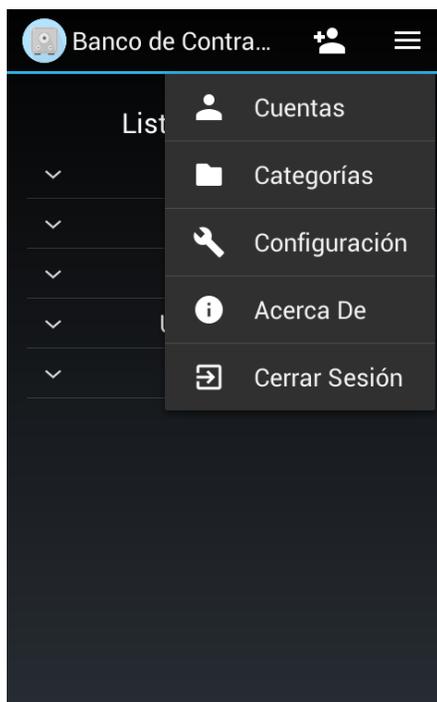
Diseño de Pantalla:

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

Nombre Lógico:

Agregar/Editar Cuenta.

Objetivo:

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

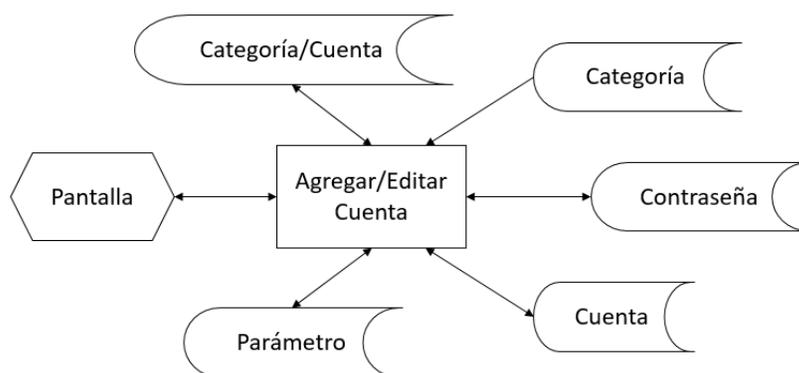
Diagrama de Bloques:

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 palabras 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:

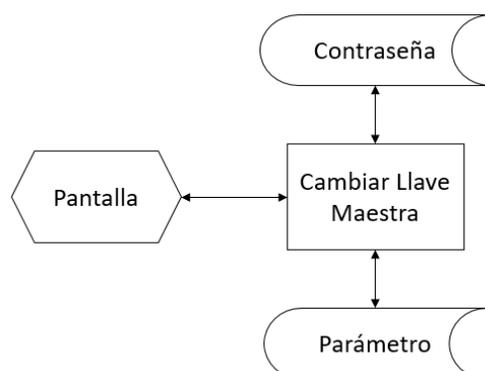


Figura 3-10. Diagrama de bloques – Cambiar Llave Maestra.

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 reencipta 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:

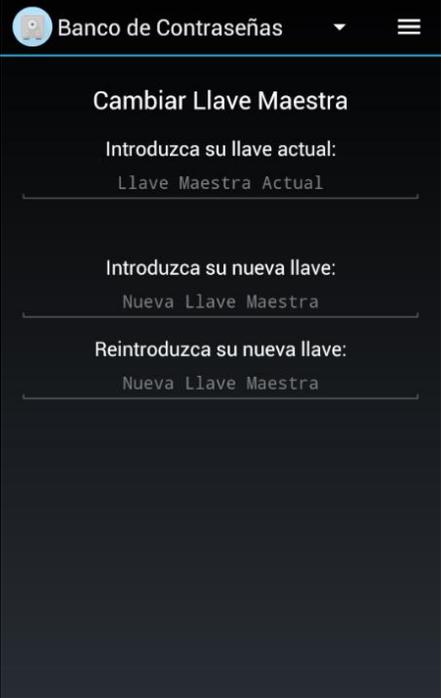
La imagen muestra una pantalla de una aplicación móvil con un fondo oscuro. En la parte superior, hay una barra de navegación con un ícono de perfil a la izquierda, el texto "Banco de Contraseñas" en el centro y un ícono de menú a la derecha. El título principal de la pantalla es "Cambiar Llave Maestra". Debajo del título, hay tres secciones de entrada de texto, cada una con un encabezado y un campo de entrada: "Introduzca su llave actual:" con el placeholder "Llave Maestra Actual", "Introduzca su nueva llave:" con el placeholder "Nueva Llave Maestra", y "Reintroduzca su nueva llave:" con el placeholder "Nueva Llave Maestra".

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

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.

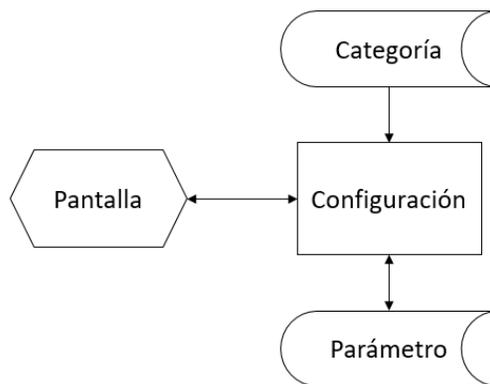
Diagrama de Bloques:

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.

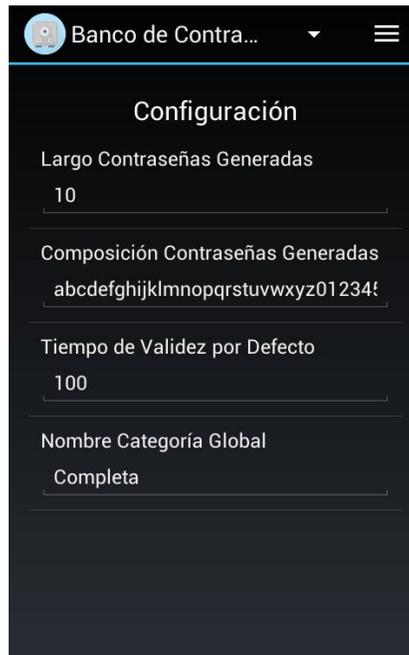
Diseño de Pantalla:

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.

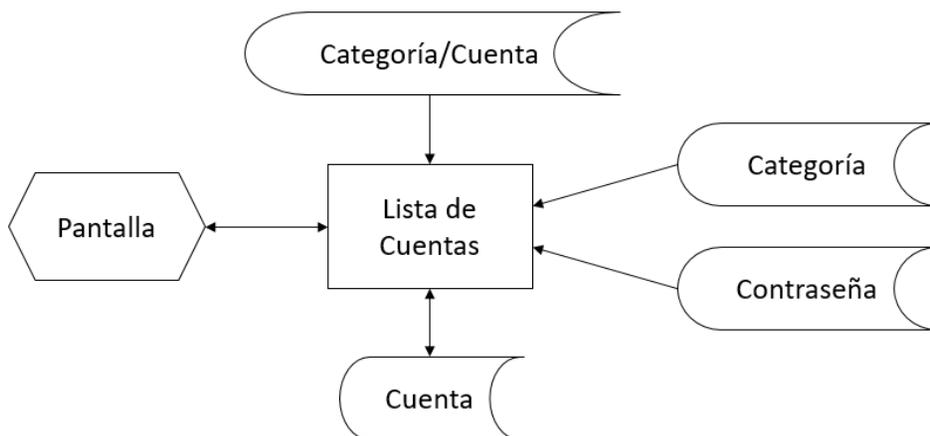
Diagrama de Bloques:

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:

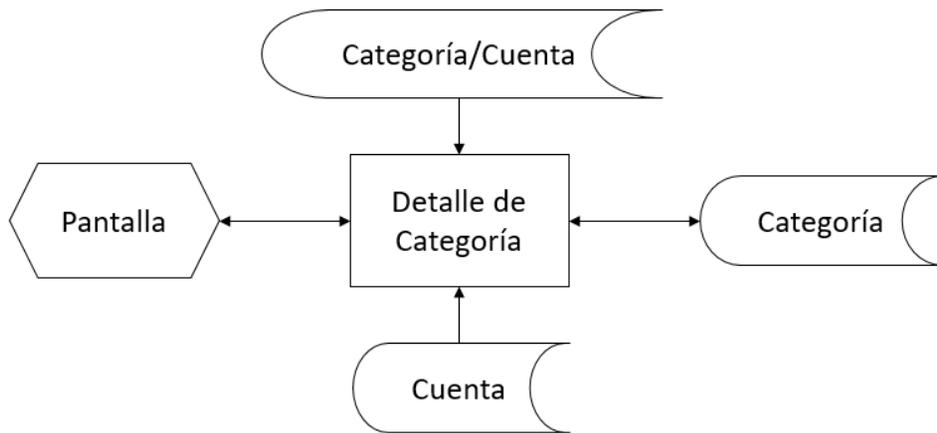


Figura 3-16. Diagrama de bloques – Detalle de Categoría.

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:

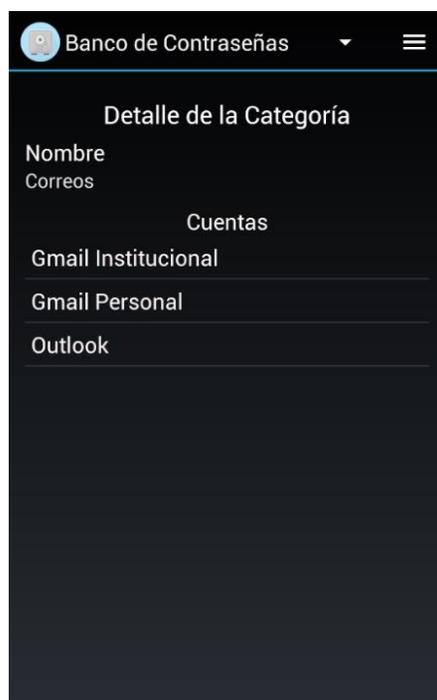


Figura 3-17. Diseño de la pantalla Detalle de Categoría.

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.

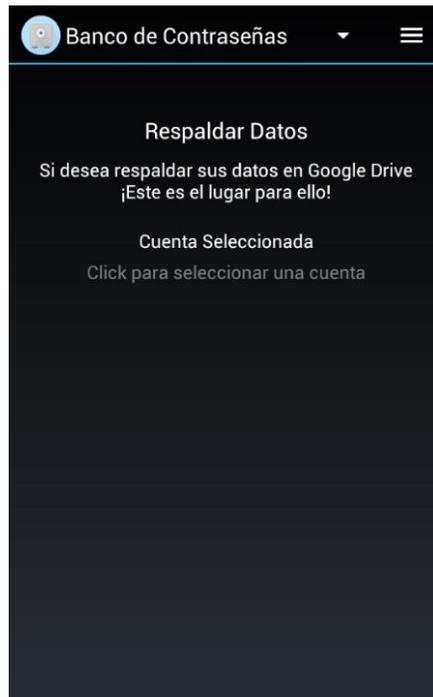
Diseño de Pantalla:

Figura 3-19. Diseño de la pantalla Respalidar 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.

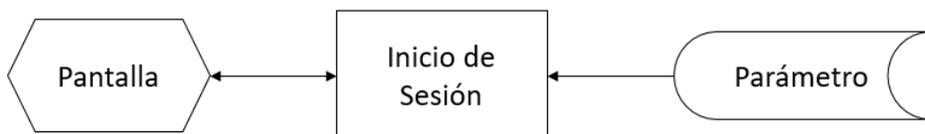
Diagrama de Bloques:

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:

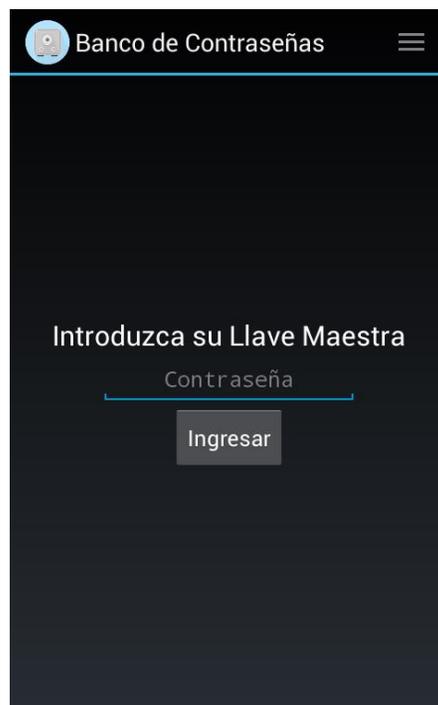


Figura 3-21. Diseño de la pantalla Inicio de Sesión.

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 =
47         getFragmentManager();
48     private Fragment fragmentoActual;
49
50     private final String estadoLogin = "Sesión Iniciada";
51     private final String onStopTime = "Tiempo Stop";
52     private final String historialFragmentos = "Historial Fragmentos";
53     private final String historialBundles = "Historial Argumentos";
54     private final String ultimoFragmento = "Último Fragmento";
55     private final String ultimoBundle = "Último Bundle";

```

```

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(
76                 estadoLogin);
77             onStop = savedInstanceState.getLong(onStopTime);
78             stringsToListaFragmentos(savedInstanceState.
79                 getStringArrayList(historialFragmentos));
80             stringsToListaBundles(savedInstanceState.
81                 getStringArrayList(historialBundles));
82             Class clase = stringToClass(savedInstanceState.getString(
83                 ultimoFragmento));
84             Bundle bundle = stringToBundle(savedInstanceState.
85                 getString(ultimoBundle));
86             try {
87                 Fragment fragmento = (Fragment) clase.getConstructor
88                     ().newInstance();
89                 fragmento.setArguments(bundle);
90                 fragmentoActual = fragmento;
91             } catch (Exception ex) {
92                 Log.e(TAG, "Error al tratar de recrear el fragmento
93                     actual", ex);
94             }
95         } else {
96             userLogout();
97         }
98         Log.i(TAG, "---" + estadoLogin + ": " + sesionIniciada);
99         Log.i(TAG, "---" + onStopTime + ": " + onStop);
100     }
101
102     @Override
103     public void onSaveInstanceState(Bundle savedInstanceState) {
104         if (onStop == null) {
105             onStop = Calendar.getInstance().getTimeInMillis();
106         }
107         Log.i(TAG, "onSaveInstanceState: ");
108         Log.i(TAG, "---" + estadoLogin + ": " + sesionIniciada);
109         Log.i(TAG, "---" + onStopTime + ": " + onStop);

```

```

103         savedInstanceState.putBoolean(estadosLogin, sesionIniciada);
104         savedInstanceState.putLong(onStopTime, onStop);
105
106         //Almacenar las listas de historiales
107         savedInstanceState.putStringArrayList(historialFragmentos,
108         listaFragmentoToStrings());
109         savedInstanceState.putStringArrayList(historialBundles,
110         listaBundlesToStrings());
111
112         //Almacenar el fragmento actual
113         savedInstanceState.putString(ultimoFragmento, classToString(
114         fragmentoActual.getClass()));
115         savedInstanceState.putString(ultimoBundle, bundleToString(
116         fragmentoActual.getArguments()));
117
118         super.onSaveInstanceState(savedInstanceState);
119     }
120
121     @Override
122     public void onActivityResult(final int requestCode, final int
123     resultCode, final Intent data) {
124         super.onActivityResult(requestCode, resultCode, data);
125         Fragment frg = getFragmentManager().findFragmentById(R.id.
126         FL_contenedor);
127         if (frg != null) {
128             frg.onActivityResult(requestCode, resultCode, data);
129         }
130     }
131
132     @Override
133     public void onStop() {
134         Log.i(TAG, "onStop():");
135         onStop = Calendar.getInstance().getTimeInMillis();
136         Log.i(TAG, "---onStop: " + onStop);
137         super.onStop();
138     }
139
140     @Override
141     public void onStart() {
142         Log.i(TAG, "onStart():");
143         if (onStop != null) {
144             long onStart = Calendar.getInstance().getTimeInMillis();
145             long diff = Math.abs(onStart - onStop);
146             Log.i(TAG, "---onStart: " + onStart + " onStop: " +
147             onStop + " Diferencia: " + diff);
148             onStop = null;
149             if (diff > waitTime) {
150                 userLogout();
151             }
152         }
153         super.onStart();
154     }
155
156     @Override
157     public void onBackPressed() {

```

```

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
193             ().newInstance();
194         fragmento.setArguments(bundle);
195
196         View view = findViewById(R.id.FL_contenedor);
197         InputMethodManager inputMethodManager = (
198             InputMethodManager) getSystemService(Context.
199             INPUT_METHOD_SERVICE);
200         inputMethodManager.hideSoftInputFromWindow(view.
201             getWindowToken(), InputMethodManager.HIDE_NOT_ALWAYS);
202
203         FragmentTransaction fragmentTransaction =
204             administradorFragmentos.beginTransaction();
205         fragmentTransaction.replace(R.id.FL_contenedor,
206             fragmento);
207         fragmentTransaction.commit();
208         fragmentoActual = fragmento;
209         System.out.println(listaFragmentos);
210     } catch (Exception ex) {
211         ex.printStackTrace();
212     }
213 }
214 //super.onBackPressed();
215 }
216
217 //Creación del menú de la aplicación
218 @Override
219 public boolean onCreateOptionsMenu(Menu menu) {
220     this.menu = menu;
221     MenuInflater inflater = getMenuInflater();
222     inflater.inflate(R.menu.menu_actividad, menu);
223     return super.onCreateOptionsMenu(menu);
224 }
225
226 //Cuando se prepare el menú de la aplicación
227 @Override
228 public boolean onPrepareOptionsMenu(Menu menu) {
229     Log.d(TAG, "onPrepareOptionsMenu executed!");
230     if (!sesionIniciada) {
231         menu.close();
232         menu.findItem(R.id.menu).setVisible(false);
233     } else {
234         menu.findItem(R.id.menu).setVisible(true);
235     }
236     return super.onPrepareOptionsMenu(menu);
237 }
238
239 //Funcionalidad del menú principal de la aplicación
240 @Override
241 public boolean onOptionsItemSelected(MenuItem item) {
242     //Si la sesión no está iniciada, no se hace nada.
243     if (!sesionIniciada) {
244         return false;
245     }
246     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)
265     getSystemService(Context.INPUT_METHOD_SERVICE);
266     inputMethodManager.hideSoftInputFromWindow(view.
267     getWindowToken(), InputMethodManager.HIDE_NOT_ALWAYS);
268
269     FragmentTransaction fragmentTransaction =
270     administradorFragmentos.beginTransaction();
271     fragmentTransaction.replace(R.id.FL_contenedor, fragmento);
272
273     if (fragmentoActual != null && fragmentoActual instanceof
274     Fragment) {
275         /*if (listaFragmentos.size() >= 10) {
276             listaFragmentos.remove(0);
277             listaBundles.remove(0);
278         }*/
279         if (listaFragmentos.size() == 0 || fragmentoActual.
280         getClass() != listaFragmentos.get(listaFragmentos.size()
281         - 1)) {
282             listaFragmentos.add(fragmentoActual.getClass());
283             listaBundles.add(fragmentoActual.getArguments());
284         }
285     }
286     if (fragmento instanceof FragInicioSesion) {
287         listaFragmentos.clear();
288         listaBundles.clear();
289     }
290
291     fragmentTransaction.commit();
292     fragmentoActual = fragmento;
293
294     Log.i(TAG, listaFragmentos.toString());
295     return true;

```

```

290     }
291
292     public void actualizarBundles(Class clase, String antiguoValor,
293     String nuevoValor) {
294         List<Class> reqCatName = new ArrayList<>();
295         reqCatName.add(FragAgregarEditarCategoria.class);
296         reqCatName.add(FragDetalleCategoria.class);
297
298         List<Class> reqCueName = new ArrayList<>();
299         reqCueName.add(FragAgregarEditarCuenta.class);
300         reqCueName.add(FragDetalleCuenta.class);
301         reqCueName.add(FragHistorialCuenta.class);
302
303         if (nuevoValor != null) {
304             Log.i(TAG, "Actualizando bundles...");
305             for (int i = 0; i < listaFragmentos.size(); i++) {
306                 Log.i(TAG, "Checkeando fragmento: " + listaFragmentos
307                 .get(i));
308                 if ((clase == Categoria.class && reqCatName.contains(
309                 listaFragmentos.get(i))) || (clase == Cuenta.class &&
310                 reqCueName.contains(listaFragmentos.get(i)))) {
311                     if (listaBundles.get(i) != null) {
312                         if ((clase == Categoria.class && listaBundles
313                         .get(i).getString(ColCategoria.NOMBRE.
314                         toString()).equals(antiguoValor)) || (clase
315                         == Cuenta.class && listaBundles.get(i).
316                         getString(ColCuenta.NOMBRE.toString()).equals
317                         (antiguoValor))) {
318                             Log.i(TAG, "Cambiando bundle...");
319                             Bundle nuevo = new Bundle();
320                             if (clase == Categoria.class) {
321                                 nuevo.putString(ColCategoria.NOMBRE.
322                                 toString(), nuevoValor);
323                             } else if (clase == Cuenta.class) {
324                                 nuevo.putString(ColCuenta.NOMBRE.
325                                 toString(), nuevoValor);
326                             }
327                             listaBundles.set(i, nuevo);
328                         }
329                     }
330                 }
331             }
332         } else {
333             Log.i(TAG, "Eliminando bundles...");
334             List<Class> nuevaListaFragmentos = new ArrayList<>(
335             listaFragmentos);
336             List<Bundle> nuevaListaBundles = new ArrayList<>(
337             listaBundles);
338             Log.i(TAG, "Limpiando lista de fragmentos...");
339             listaFragmentos.clear();
340             listaBundles.clear();
341             for (int i = 0; i < nuevaListaFragmentos.size(); i++) {
342                 Log.i(TAG, "Checkeando fragmento: " +
343                 nuevaListaFragmentos.get(i));

```

```

330     if ((clase == Categoria.class && reqCatName.contains(
331         nuevaListaFragmentos.get(i))) || (clase == Cuenta.
332         class && reqCueName.contains(nuevaListaFragmentos.get
333         (i)))) {
334         if (nuevaListaBundles.get(i) != null) {
335             if ((clase == Categoria.class &&
336                 nuevaListaBundles.get(i).getString(
337                 ColCategoria.NOMBRE.toString()).equals(
338                 antiguoValor)) || (clase == Cuenta.class &&
339                 nuevaListaBundles.get(i).getString(ColCuenta.
340                 NOMBRE.toString()).equals(antiguoValor))) {
341                 Log.i(TAG, "Descartando fragmento: " +
342                     nuevaListaFragmentos.get(i));
343             } else {
344                 if (listaFragmentos.size() == 0 ||
345                     nuevaListaFragmentos.get(i) !=
346                     listaFragmentos.get(listaFragmentos.size
347                     () - 1)) {
348                     Log.i(TAG, "Volviendo a agregar
349                     fragmento: " + nuevaListaFragmentos.
350                     get(i));
351                     listaFragmentos.add(
352                     nuevaListaFragmentos.get(i));
353                     listaBundles.add(nuevaListaBundles.
354                     get(i));
355                 } else {
356                     Log.i(TAG, "Descartando fragmento: "
357                         + nuevaListaFragmentos.get(i));
358                 }
359             }
360         } else {
361             if (listaFragmentos.size() == 0 ||
362                 nuevaListaFragmentos.get(i) !=
363                 listaFragmentos.get(listaFragmentos.size() -
364                 1)) {
365                 Log.i(TAG, "Volviendo a agregar
366                 fragmento: " + nuevaListaFragmentos.get(i
367                 ));
368                 listaFragmentos.add(nuevaListaFragmentos.
369                 get(i));
370                 listaBundles.add(nuevaListaBundles.get(i
371                 ));
372             } else {
373                 Log.i(TAG, "Descartando fragmento: " +
374                     nuevaListaFragmentos.get(i));
375             }
376         }
377     } else {
378         if (listaFragmentos.size() == 0 ||
379             nuevaListaFragmentos.get(i) != listaFragmentos.
380             get(listaFragmentos.size() - 1)) {
381             Log.i(TAG, "Volviendo a agregar fragmento: "
382                 + nuevaListaFragmentos.get(i));
383             listaFragmentos.add(nuevaListaFragmentos.get(
384             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     {
422         return Categoria.class.getName() + ":" + bundle.getString(
423             ColCategoria.NOMBRE.toString());
424     }
425     if (bundle.getString(ColCuenta.NOMBRE.toString()) != null) {
426         return Cuenta.class.getName() + ":" + bundle.getString(
427             ColCuenta.NOMBRE.toString());
428     }
429     return null;
430 }
431
432 //Se carga la lista de strings en el historial de fragmentos
433 private void stringsToListaFragmentos(ArrayList<String> entrada) {
434     Log.i(TAG, entrada.toString());
435     listaFragmentos.clear();
436     for (String nombreFrag : entrada) {
437         listaFragmentos.add(stringToClass(nombreFrag));
438     }
439 }
440
441 private Class stringToClass(String nombre) {
442     if (FragAcercaDe.class.getName().equals(nombre)) {
443         return FragAcercaDe.class;
444     }
445     if (FragAgregarEditarCategoria.class.getName().equals(nombre)) {
446         return FragAgregarEditarCategoria.class;
447     }
448     if (FragAgregarEditarCuenta.class.getName().equals(nombre)) {
449         return FragAgregarEditarCuenta.class;
450     }
451     if (FragCambioLlaveMaestra.class.getName().equals(nombre)) {
452         return FragCambioLlaveMaestra.class;
453     }
454     if (FragCategorias.class.getName().equals(nombre)) {
455         return FragCategorias.class;
456     }
457     if (FragConfiguracion.class.getName().equals(nombre)) {
458         return FragConfiguracion.class;
459     }
460     if (FragCrearLlaveMaestra.class.getName().equals(nombre)) {
461         return FragCrearLlaveMaestra.class;
462     }
463 }

```

```

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 }

```

```

1  package cl.theroot.passbank.fragmento;
2
3  import android.app.AlertDialog;
4  import android.app.Fragment;
5  import android.content.DialogInterface;
6  import android.os.Bundle;
7  import android.text.Editable;
8  import android.text.TextWatcher;
9  import android.view.ContextMenu;
10 import android.view.LayoutInflater;
11 import android.view.Menu;
12 import android.view.MenuInflater;
13 import android.view.MenuItem;
14 import android.view.MotionEvent;
15 import android.view.View;
16 import android.view.ViewGroup;
17 import android.widget.EditText;
18 import android.widget.ImageView;
19 import android.widget.ListAdapter;
20 import android.widget.ListView;
21 import android.widget.SeekBar;
22 import android.widget.TextView;
23
24 import java.util.ArrayList;
25 import java.util.Calendar;
26 import java.text.SimpleDateFormat;
27 import java.util.List;
28
29 import cl.theroot.passbank.ActividadPrincipal;
30 import cl.theroot.passbank.Cifrador;
31 import cl.theroot.passbank.CustomToast;
32 import cl.theroot.passbank.GeneradorContrasennas;
33 import cl.theroot.passbank.adaptador.AdapCategoriasCheckBox;
34 import cl.theroot.passbank.ExcepcionBancoContrasennas;
35 import cl.theroot.passbank.R;
36 import cl.theroot.passbank.dominio.Categoria;
37 import cl.theroot.passbank.dominio.CategoriaCuenta;
38 import cl.theroot.passbank.dominio.CategoriaSeleccionable;
39 import cl.theroot.passbank.dominio.Contrasenna;
40 import cl.theroot.passbank.dominio.Cuenta;
41 import cl.theroot.passbank.dominio.Parametro;
42 import cl.theroot.passbank.datos.CategoriaCuentaDAO;
43 import cl.theroot.passbank.datos.CategoriaDAO;
44 import cl.theroot.passbank.datos.ContrasennaDAO;
45 import cl.theroot.passbank.datos.CuentaDAO;
46 import cl.theroot.passbank.datos.ParametroDAO;
47 import cl.theroot.passbank.datos.nombres.ColCuenta;
48 import cl.theroot.passbank.datos.nombres.NombreParametro;
49
50 public class FragAgregarEditarCuenta extends Fragment implements View
51     .OnClickListener{
52     private AdapCategoriasCheckBox adapter;
53     private List<CategoriaSeleccionable> categories;
54     private EditText ET_name;

```

```

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_titulo = view.findViewById(R.id.TV_titulo);
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.
103     IV_passGenerator);
104     IV_passGenerator.setOnClickListener(this);
105     registerForContextMenu(IV_passGenerator);
106
107     ListView listView = view.findViewById(R.id.
108     listview_categories_checkboxes);
109     listView.setOnTouchListener(new View.OnTouchListener() {
110         @Override
111         public boolean onTouch(View view, MotionEvent motionEvent
112         ) {
113             view.getParent().requestDisallowInterceptTouchEvent(
114             true);
115             return false;
116         }
117     });
118
119     final SeekBar SB_validez = view.findViewById(R.id.SB_validez);
120     SB_validez.setOnSeekBarChangeListener(new SeekBar.
121     OnSeekBarChangeListener() {
122         @Override
123         public void onProgressChanged(SeekBar seekBar, int
124         progress, boolean fromUser) {
125             if (fromUser) {
126                 ET_validez.setText(String.valueOf(progress));
127             }
128         }
129
130         @Override
131         public void onStartTrackingTouch(SeekBar seekBar) {
132             }
133
134         @Override
135         public void onStopTrackingTouch(SeekBar seekBar) {
136             }
137     });
138
139     ET_validez.addTextChangedListener(new TextWatcher() {
140         @Override
141         public void beforeTextChanged(CharSequence s, int start,
142         int count, int after) {
143             }
144
145         @Override
146         public void onTextChanged(CharSequence s, int start, int
147         before, int count) {
148             }
149
150         @Override
151         public void afterTextChanged(Editable s) {

```

```

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

```

```

198         oldName = null;
199     }
200 }
201 }
202
203 fillCategoriesInfo(oldName);
204 adapter = new AdapCategoriasCheckBox(getActivity(),
205     categories, this);
206 listView.setAdapter(adapter);
207 setListViewHeightBasedOnChildren(listView);
208
209 if (categories.isEmpty()) {
210     TV_subTitulo.setVisibility(View.INVISIBLE);
211 }
212
213 //Setear el modo del generador por defecto
214 Parametro parametro = parametroDAO.seleccionarUno(
215     NombreParametro.ULTIMO_MODO_GENERADOR.toString());
216 if (parametro != null) {
217     modoGenerador = Byte.parseByte(parametro.getValor());
218 }
219
220 //Setear datos para habilitar/deshabilitar el botón guardar
221 algunCambio = false;
222 nombreOriginal = ET_name.getText().toString();
223 ET_name.addTextChangedListener(new TextWatcher() {
224     @Override
225     public void beforeTextChanged(CharSequence s, int start,
226         int count, int after) {
227     }
228     @Override
229     public void onTextChanged(CharSequence s, int start, int
230         before, int count) {
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,
241         int count, int after) {
242     }
243     @Override
244     public void onTextChanged(CharSequence s, int start, int
245         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,
258         int count, int after) {
259
260     }
261
262     @Override
263     public void onTextChanged(CharSequence s, int start, int
264         before, int count) {
265
266     }
267
268     @Override
269     public void afterTextChanged(Editable s) {
270         checkearCambios();
271     }
272 });
273 validezOriginal = ET_validez.getText().toString();
274 ET_validez.addTextChangedListener(new TextWatcher() {
275     @Override
276     public void beforeTextChanged(CharSequence s, int start,
277         int count, int after) {
278
279     }
280
281     @Override
282     public void onTextChanged(CharSequence s, int start, int
283         before, int count) {
284
285     }
286
287     @Override
288     public void afterTextChanged(Editable s) {
289         checkearCambios();
290     }
291 });
292
293 return view;
294 }
295
296 //Creación del submenu del fragmento
297 @Override
298 public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
299 {
300     inflater.inflate(R.menu.sub_menu_agregar_editar_cuenta, menu);
301     super.onCreateOptionsMenu(menu, inflater);

```

```

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
Contrasenna contAntigua = contrasennaDAO.
seleccionarUna(oldPasswordID);
375

```

```

376 
377 
378
379
380
381
382
383
384
385
386
387
388
389
390
391 
392
393
394 
395
396 
397
398
399
400
401 
402
403
404
405
406
407 

```

```

    if (contAntigua != null) {
        if (!Cifrador.desencriptar(contAntigua.
            getValor(), ActividadPrincipal.
            llaveEncrip).equals(password)) {
            Contrasenna contrasenna = new
            Contrasenna(null, oldName, Cifrador.
            encriptar(password,
            ActividadPrincipal.llaveEncrip),
            formattedDate);
            oldPasswordID = contrasennaDAO.
            insertarUna(contrasenna);
        }
    } else {
        Contrasenna contrasenna = new Contrasenna
        (null, oldName, Cifrador.encriptar(
        password, ActividadPrincipal.llaveEncrip
        ), formattedDate);
        oldPasswordID = contrasennaDAO.
        insertarUna(contrasenna);
    }
} else {
    Contrasenna contrasenna = new Contrasenna(
    null, oldName, Cifrador.encriptar(password,
    ActividadPrincipal.llaveEncrip),
    formattedDate);
    oldPasswordID = contrasennaDAO.insertarUna(
    contrasenna);
}

//Ya creada/actualizada la cuenta y contraseña,
//se deben actualizar las relaciones entre
//categorias y cuentas.
for (CategoriaSeleccionable categoria : adapter.
getCategorias()) {
    //Checkear si existe la relación
    CategoriaCuenta categoriaCuenta =
    categoriaCuentaDAO.seleccionarUna(categoria.
    getNombre(), oldName);
    if (categoria.isSeleccionado()) {
        //Si no existe, la creamos.
        if (categoriaCuenta == null) {
            categoriaCuentaDAO.insertarUna(new
            CategoriaCuenta(categoria.getNombre
            (), oldName, 0));
        }
    } else {
        //Si existe, la eliminamos.
        if (categoriaCuenta != null) {
            categoriaCuentaDAO.eliminarUna(
            categoria.getNombre(), oldName);
        }
    }
}

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     {
455         return false;
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.
464             SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW);
465             item.setActionView(new View(getActivity().
466             getApplicationContext()));
467             item.setOnActionExpandListener(new MenuItem.
468             OnActionExpandListener() {
469                 @Override
470                 public boolean onMenuItemActionExpand(MenuItem
471                 item) {
472                     return false;
473                 }
474             });
475
476             return false;
477         case R.id.cont_menu_agregar_editar_cuenta_palabras:
478             item.setChecked(true);
479             modoGenerador = 1;
480
481             //Para mantener abierto el context menu al hacer click
482             item.setShowAsAction(MenuItem.
483             SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW);
484             item.setActionView(new View(getActivity().
485             getApplicationContext()));
486             item.setOnActionExpandListener(new MenuItem.
487             OnActionExpandListener() {
488                 @Override
489                 public boolean onMenuItemActionExpand(MenuItem
490                 item) {
491                     return false;
492                 }
493             });
494
495             return false;
496         default:
497             return false;
498     }
499 }
500 @Override

```

```

492     public boolean onOptionsItemSelected (MenuItem
493     item) {
494         return false;
495     }
496     });
497     return false;
498     default:
499     return super.onOptionsItemSelected(item);
500     }
501     }
502
503     //Funcionalidad del botón para hacer visible/invisible la
504     contraseña
505     @Override
506     public void onClick(View view) {
507         switch (view.getId()) {
508             case R.id.IV_passVisibility:
509                 if (ET_password.getInputType() == 0x00000081) {
510                     int pointerPos = ET_password.getSelectionStart();
511                     IV_passVisibility.setImageResource(R.drawable.
512                     ic_visibility_off_white_24dp);
513                     ET_password.setInputType(0x00080001);
514                     ET_password.setSelection(pointerPos);
515                 }
516                 else {
517                     int pointerPos = ET_password.getSelectionStart();
518                     IV_passVisibility.setImageResource(R.drawable.
519                     ic_visibility_white_24dp);
520                     ET_password.setInputType(0x00000081);
521                     ET_password.setSelection(pointerPos);
522                 }
523                 break;
524             case R.id.IV_passGenerator:
525                 parametroDAO.actualizarUna(new Parametro(
526                 NombreParametro.ULTIMO_MODO_GENERADOR.toString(),
527                 String.valueOf(modosGenerador), null));
528                 if (modosGenerador == 0) {
529                     ET_password.setText(genContrasennas.generar(true
530                     ));
531                 } else {
532                     ET_password.setText(genContrasennas.generar(false
533                     ));
534                 }
535                 break;
536         }
537     }
538
539     private void fillCategoriesInfo(String nombreCuenta) {
540         categories = new ArrayList<>();
541
542         for (Categoria categoria : categoriaDAO.seleccionarTodas()) {
543             CategoriaSeleccionable categoriaSeleccionable = new
544             CategoriaSeleccionable(categoria.getNombre(), categoria.
545             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(
586         validezOriginal)) {
587         algunCambio = true;
588         getActivity().invalidateOptionsMenu();
589         return;
590     }
591     if (adapter.checkearCambios()) {
592         algunCambio = true;
593         getActivity().invalidateOptionsMenu();
594         return;
595     }
596     getActivity().invalidateOptionsMenu();
597 }
598 }
```

```

1  package cl.theroot.passbank.fragmento;
2
3
4  import android.app.AlertDialog;
5  import android.app.Fragment;
6  import android.content.DialogInterface;
7  import android.os.Bundle;
8  import android.text.Editable;
9  import android.text.TextWatcher;
10 import android.view.LayoutInflater;
11 import android.view.Menu;
12 import android.view.MenuInflater;
13 import android.view.MenuItem;
14 import android.view.View;
15 import android.view.ViewGroup;
16 import android.widget.EditText;
17
18 import cl.theroot.passbank.ActividadPrincipal;
19 import cl.theroot.passbank.Cifrador;
20 import cl.theroot.passbank.CustomToast;
21 import cl.theroot.passbank.ExcepcionBancoContrasennas;
22 import cl.theroot.passbank.R;
23 import cl.theroot.passbank.dominio.Contrasenna;
24 import cl.theroot.passbank.dominio.Parametro;
25 import cl.theroot.passbank.datos.ContrasennaDAO;
26 import cl.theroot.passbank.datos.ParametroDAO;
27 import cl.theroot.passbank.datos.nombres.NombreParametro;
28
29 /**
30  * A simple {@link Fragment} subclass.
31  */
32 public class FragCambioLlaveMaestra extends Fragment{
33     private static final String TAG = "BdC-FragCambioLlaveMaestra";
34     private EditText ET_oldPassword;
35     private EditText ET_newPassword;
36     private EditText ET_newRepPassword;
37     private ParametroDAO parametroDAO;
38     private boolean habilitarCambios = false;
39
40     @Override
41     public View onCreateView(LayoutInflater inflater, ViewGroup
42     container, Bundle savedInstanceState) {
43         setHasOptionsMenu(true);
44         View view = inflater.inflate(R.layout.
45         fragmento_cambio_llave_maestra, null);
46         ET_oldPassword = view.findViewById(R.id.ET_oldPassword);
47         ET_newPassword = view.findViewById(R.id.ET_newPassword);
48         ET_newRepPassword = view.findViewById(R.id.ET_newRepPassword);
49         ET_oldPassword.addTextChangedListener(new TextWatcher() {
50             @Override
51             public void beforeTextChanged(CharSequence s, int start,
52             int count, int after) {

```

```

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

```

```

101 public void onCreateOptionsMenu(Menu menu, MenuInflater
102 menuInflater) {
103     menuInflater.inflate(R.menu.sub_menu_cambio_llave_maestra,
104     menu);
105     super.onCreateOptionsMenu(menu, menuInflater);
106 }
107
108 @Override
109 public void onPrepareOptionsMenu(Menu menu) {
110     menu.findItem(R.id.sub_menu_cambio_llave_maestra_guardar).
111     setEnabled(habilitarCambios);
112     super.onPrepareOptionsMenu(menu);
113 }
114
115 @Override
116 public boolean onOptionsItemSelected(MenuItem menuItem) {
117     if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
118     {
119         return false;
120     }
121     switch(menuItem.getItemId()) {
122     case R.id.sub_menu_cambio_llave_maestra_volver:
123         getActivity().onBackPressed();
124         return true;
125     case R.id.sub_menu_cambio_llave_maestra_guardar:
126         try {
127             //Procesar cambio de contraseña
128             String contVieja = ET_oldPassword.getText().
129             toString();
130             String contNueva = ET_newPassword.getText().
131             toString();
132             String contNuevaConf = ET_newRepPassword.getText
133             ().toString();
134
135             //Validar contraseña vieja
136             if (contVieja.isEmpty()) {
137                 throw new ExcepcionBancoContrasennas("Error
138                 - Llave Maestra Requerida", "Se requiere el
139                 ingreso de su Llave Maestra Actual para
140                 realizar el cambio");
141             }
142             Parametro parSalt = parametroDAO.seleccionarUno(
143             NombreParametro.SAL_HASH.toString());
144             Parametro parHash = parametroDAO.seleccionarUno(
145             NombreParametro.RESULTADO_HASH.toString());
146             String[] saltYHashObt = Cifrador.genHashedPass(
147             contVieja, parSalt.getValor());
148             if (!saltYHashObt[1].equals(parHash.getValor())) {
149                 throw new ExcepcionBancoContrasennas("Error
150                 - Llave Maestra Incorrecta", "La Llave
151                 Maestra Actual ingresada no es correcta,
152                 favor intentar de nuevo");
153             }
154         }
155     }
156 }

```

```

139 //Validar contraseña nueva
140 if (contNueva.isEmpty()) {
141     throw new ExcepcionBancoContrasennas("Error
        - Campo Vacio", "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, deberia 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().
168             getApplicationContext(), "Su Llave
169             Maestra fue actualizada exitosadamente"
170         );
171         ((ActividadPrincipal) getActivity()).
172         cambiarFragmento(new
173             FragConfiguracion());
174     }
175 }
176 } catch(ExcepcionBancoContrasennas ex) {
177     AlertDialog alertDialog = new AlertDialog.Builder
178     (getActivity()).create();
179     alertDialog.setTitle(ex.getTitulo());
180     alertDialog.setMessage(ex.getMensaje());
181     alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL,
182         "OK", new DialogInterface.OnClickListener() {
183         public void onClick(DialogInterface dialog,
184             int which) {
185             dialog.dismiss();
186         }
187     });
188     alertDialog.show();
189 }
190 return true;
191 default:
192     return super.onOptionsItemSelected(menuItem);
193 }
194 }
195
196 private void reencriptarContrasennas(String llaveEncrVieja,
197     String llaveEncrNueva) {
198     ContrasennaDAO contrasennaDAO = new ContrasennaDAO(
199     getActivity().getApplicationContext());
200     for (Contrasenna contrasenna : contrasennaDAO.
201     seleccionarTodas()) {
202         String valorDesencriptado = Cifrador.desencriptar(
203         contrasenna.getValor(), llaveEncrVieja);
204         String valorEncriptado = Cifrador.encriptar(
205         valorDesencriptado, llaveEncrNueva);
206         contrasenna.setValor(valorEncriptado);
207         contrasennaDAO.actualizarUna(contrasenna);
208     }
209 }
210
211 private void habilitarCambios() {
212     habilitarCambios = true;
213     if (ET_oldPassword.getText().toString().isEmpty()) {
214         habilitarCambios = false;
215     }
216     if (ET_newPassword.getText().toString().isEmpty()) {
217         habilitarCambios = false;
218     }
219     if (ET_newRepPassword.getText().toString().isEmpty()) {
220         habilitarCambios = false;
221     }
222     getActivity().invalidateOptionsMenu();
223 }

```

```

1  package cl.theroot.passbank.fragmento;
2
3
4  import android.app.AlertDialog;
5  import android.app.Fragment;
6  import android.content.DialogInterface;
7  import android.os.Bundle;
8  import android.util.Log;
9  import android.view.LayoutInflater;
10 import android.view.Menu;
11 import android.view.MenuInflater;
12 import android.view.MenuItem;
13 import android.view.View;
14 import android.view.ViewGroup;
15 import android.widget.ListView;
16
17 import java.util.ArrayList;
18 import java.util.Arrays;
19 import java.util.List;
20
21 import cl.theroot.passbank.ActividadPrincipal;
22 import cl.theroot.passbank.adaptador.AdapParametros;
23 import cl.theroot.passbank.CustomToast;
24 import cl.theroot.passbank.R;
25 import cl.theroot.passbank.dominio.Parametro;
26 import cl.theroot.passbank.dominio.ParametroSeleccionable;
27 import cl.theroot.passbank.datos.CategoriaDAO;
28 import cl.theroot.passbank.datos.ParametroDAO;
29 import cl.theroot.passbank.datos.nombres.NombreParametro;
30
31
32 public class FragConfiguracion extends Fragment {
33     private static final String TAG = "BdC-FragConfiguracion";
34     private AdapParametros adapter;
35     private List<ParametroSeleccionable> parametros;
36
37     public static final String[] PARAMETROS_NUMERICOS = {
38         NombreParametro.CANT_PALABRAS_GENERADOR.toString(),
39         NombreParametro.CANT_CARACTERES_GENERADOR.toString(),
40         NombreParametro.VALIDEZ_DEFECTO.toString()
41     };
42
43     public static final String[] PARAMETROS_NO_TRIMEABLES = {
44         NombreParametro.SEPARADOR_GENERADOR.toString(),
45         NombreParametro.COMPOSICION_GENERADOR.toString()
46     };
47
48     public static final String[] PARAMETROS_NO_REPETIBLES = {
49         NombreParametro.COMPOSICION_GENERADOR.toString()
50     };
51
52     private boolean parametrosCambiados = false;
53
54     private CategoriaDAO categoriaDAO;
55     private ParametroDAO parametroDAO;

```

```

56
57
58 public View onCreateView(LayoutInflater inflater, ViewGroup
59     container, Bundle savedInstanceState) {
60     setHasOptionsMenu(true);
61     parametroDAO = new ParametroDAO(getActivity().
62         getApplicationContext());
63     categoriaDAO = new CategoriaDAO(getActivity().
64         getApplicationContext());
65
66     View view = inflater.inflate(R.layout.fragmento_configuracion
67         , container, false);
68
69     ListView listView = view.findViewById(R.id.listConfiguration);
70
71     parametros = new ArrayList<>();
72     for (Parametro parametro : parametroDAO.seleccionarVisibles
73         ()) {
74         parametros.add(new ParametroSeleccionable(parametro.
75             getNombre(), parametro.getValor(), parametro.getPosicion
76             (), false));
77     }
78     adapter = new AdapParametros(getActivity(), parametros, this);
79     listView.setAdapter(adapter);
80
81     return view;
82 }
83
84 //Creación del submenu del fragmento
85 @Override
86 public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
87 {
88     Log.d(TAG, "onCreateOptionsMenu executed!");
89     inflater.inflate(R.menu.sub_menu_configuracion, menu);
90     super.onCreateOptionsMenu(menu, inflater);
91 }
92
93 @Override
94 public void onPrepareOptionsMenu(Menu menu) {
95     Log.d(TAG, "onPrepareOptionsMenu executed!");
96     menu.findItem(R.id.sub_menu_configuracion_save).setEnabled(
97         parametrosCambiados);
98     super.onPrepareOptionsMenu(menu);
99 }
100
101 //Creación de la funcionalidad del submenu del fragmento
102 @Override
103 public boolean onOptionsItemSelected(MenuItem item) {
104     if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
105     {
106         return false;
107     }
108     switch (item.getItemId()) {
109         case R.id.sub_menu_configuracion_change_master_key:

```

```

100     return ((ActividadPrincipal) getActivity()).
101           cambiarFragmento(new FragCambioLlaveMaestra());
102     case R.id.sub_menu_configuration_export_database:
103     return ((ActividadPrincipal) getActivity()).
104           cambiarFragmento(new FragExportar());
105     case R.id.sub_menu_configuration_save:
106     if (!parametrosCambiados) {
107         new CustomToast(getActivity().
108             getApplicationContext(), "¡Acción Cancelada!\nNo
109             hay cambios que se deban almacenar.");
110         return true;
111     }
112     parametros = adapter.getParametros();
113     String salida = parametrosValidos(parametros);
114     if (salida == null) {
115         AlertDialog alertDialog = new AlertDialog.Builder
116             (getActivity()).create();
117         alertDialog.setTitle("Configuración Actualizada");
118         alertDialog.setMessage("Los cambios ejecutados
119             fueron guardados exitosamente.");
120         alertDialog.setButton(AlertDialog.BUTTON_POSITIVE
121             , "OK", new DialogInterface.OnClickListener() {
122                 @Override
123                 public void onClick(DialogInterface dialog,
124                     int which) {
125                     dialog.dismiss();
126                 }
127             });
128         alertDialog.show();
129
130     for (Parametro parametro : parametros) {
131         parametroDAO.actualizarUna(parametro);
132     }
133
134     parametros = new ArrayList<>();
135     for (Parametro parametro : parametroDAO.
136         seleccionarVisibles()) {
137         parametros.add(new ParametroSeleccionable(
138             parametro.getNombre(), parametro.getValor(),
139             parametro.getPosicion(), false));
140     }
141     adapter.updateParametros(parametros);
142     return true;
143 } else {
144     AlertDialog alertDialog = new AlertDialog.Builder
145         (getActivity()).create();
146     alertDialog.setTitle("Error, Datos Inválidos");
147     alertDialog.setMessage(salida);
148     alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL,
149         "OK", new DialogInterface.OnClickListener() {
150             @Override
151             public void onClick(DialogInterface dialog,
152                 int which) {
153                 dialog.dismiss();
154             }
155         });

```

```

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>
151     parametros) {
152     for (Parametro parametro : parametros){
153         if (parametro.getValor() == null) {
154             return parametro.getNombre() + "\nNo puede ser nulo.";
155         }
156
157         if (parametro.getValor().length() == 0) {
158             return parametro.getNombre() + "\nNo puede estar
159             vacío.";
160         }
161
162         if (!Arrays.asList(FragConfiguracion.
163             PARAMETROS_NO_TRIMEABLES).contains(parametro.getNombre
164             ())) {
165             if (parametro.getValor().trim().isEmpty()) {
166                 return parametro.getNombre() + "\nNo puede
167                 estar vacío.";
168             }
169         }
170
171         if (Arrays.asList(PARAMETROS_NUMERICOS).contains(
172             parametro.getNombre())) {
173             try {
174                 Integer.parseInt(parametro.getValor());
175             } catch(NumberFormatException ex) {
176                 return parametro.getNombre() + "\nDebe ser
177                 numérico.";
178             }
179         }
180
181         if (parametro.getNombre().equals(NombreParametro.
182             NOMBRE_CATEGORIA_COMPLETA.toString())) {
183             if (categoriaDAO.seleccionarUna(parametro.getValor())
184                 != null) {
185                 return parametro.getNombre() + "\nDebe tener un
186                 nombre distinto al resto de categorías.";
187             }
188         }
189     }
190     return null;
191 }
192
193 public void habilitarCambios(Boolean estado) {
194     if (parametrosCambiados != estado) {
195         parametrosCambiados = estado;
196         getActivity().invalidateOptionsMenu();
197     }
198 }
199
200 }

```

```

1  package cl.theroot.passbank.fragmento;
2
3  import android.app.AlertDialog;
4  import android.app.Fragment;
5  import android.content.DialogInterface;
6  import android.os.Bundle;
7  import android.util.Log;
8  import android.view.ContextMenu;
9  import android.view.LayoutInflater;
10 import android.view.Menu;
11 import android.view.MenuInflater;
12 import android.view.MenuItem;
13 import android.view.View;
14 import android.view.ViewGroup;
15 import android.widget.ExpandableListView;
16
17 import java.util.ArrayList;
18 import java.util.List;
19
20 import cl.theroot.passbank.ActividadPrincipal;
21 import cl.theroot.passbank.adaptador.AdapCategoriasCuentas;
22 import cl.theroot.passbank.R;
23 import cl.theroot.passbank.dominio.Categoria;
24 import cl.theroot.passbank.dominio.CategoriaCuenta;
25 import cl.theroot.passbank.dominio.CategoriaListaCuentas;
26 import cl.theroot.passbank.dominio.Contrasenna;
27 import cl.theroot.passbank.dominio.Cuenta;
28 import cl.theroot.passbank.dominio.CuentaConFecha;
29 import cl.theroot.passbank.datos.CategoriaCuentaDAO;
30 import cl.theroot.passbank.datos.CategoriaDAO;
31 import cl.theroot.passbank.datos.ContrasennaDAO;
32 import cl.theroot.passbank.datos.CuentaDAO;
33 import cl.theroot.passbank.datos.nombres.ColCuenta;
34
35 public class FragCuentas extends Fragment {
36     private static final String TAG = "BdC-AdapCuentas";
37     private AdapCategoriasCuentas adapter;
38
39     private CuentaDAO cuentaDAO;
40     private CategoriaDAO categoriaDAO;
41     private CategoriaCuentaDAO categoriaCuentaDAO;
42     private ContrasennaDAO contrasennaDAO;
43
44     private List<CategoriaListaCuentas> listaCategorias = new
45     ArrayList<>();
46
47     public Cuenta selectedAccount;
48
49     public FragCuentas() {
50     }
51
52     @Override
53     public View onCreateView(LayoutInflater inflater, ViewGroup
54     container, Bundle savedInstanceState) {

```

```

54     setHasOptionsMenu(true);
55     cuentaDAO = new CuentaDAO(getActivity().getApplicationContext
56     ());
57     categoriaDAO = new CategoriaDAO(getActivity().
58     getApplicationContext());
59     categoriaCuentaDAO = new CategoriaCuentaDAO(getActivity().
60     getApplicationContext());
61     contrasennaDAO = new ContrasennaDAO(getActivity().
62     getApplicationContext());
63
64     View view = inflater.inflate(R.layout.fragmento_cuentas, null
65     );
66     ExpandableListView expListView = view.findViewById(R.id.
67     expListView);
68
69     fillAccountsInfo();
70
71     adapter = new AdapCategoriasCuentas(getActivity(),
72     listaCategorias);
73     expListView.setAdapter(adapter);
74     registerForContextMenu(expListView);
75
76     expListView.setOnGroupClickListener(new ExpandableListView.
77     OnGroupClickListener() {
78         @Override
79         public boolean onGroupClick(ExpandableListView
80         expandableListView, View view, int i, long l) {
81             return false;
82         }
83     });
84
85     expListView.setOnGroupExpandListener(new ExpandableListView.
86     OnGroupExpandListener() {
87         @Override
88         public void onGroupExpand(int i) {
89
90         }
91     });
92
93     expListView.setOnGroupCollapseListener(new ExpandableListView
94     .OnGroupCollapseListener() {
95         @Override
96         public void onGroupCollapse(int i) {
97
98         }
99     });
100
101     expListView.setOnChildClickListener(new ExpandableListView.
102     OnChildClickListener() {
103         @Override
104         public boolean onChildClick(ExpandableListView
105         expandableListView, View view, int i, int il, long l) {
106             Cuenta cuenta = (Cuenta) adapter.getChild(i, il);
107             FragDetalleCuenta fragDetalleCuenta = new
108             FragDetalleCuenta();

```

```

95         Bundle bundle = new Bundle();
96         bundle.putString(ColCuenta.NOMBRE.toString(), cuenta.
97             getNombre());
98         fragDetalleCuenta.setArguments(bundle);
99         return ((ActividadPrincipal) getActivity()).
100             cambiarFragmento(fragDetalleCuenta);
101     }
102 }
103
104
105
106 //Creación del submenu principal del fragmento
107 @Override
108 public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
109 {
110     inflater.inflate(R.menu.sub_menu_cuentas, menu);
111     super.onCreateOptionsMenu(menu, inflater);
112 }
113
114 //Creación de la funcionalidad del submenu del fragmento
115 public boolean onOptionsItemSelected(MenuItem item) {
116     if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
117     {
118         return false;
119     }
120     switch (item.getItemId()) {
121         case R.id.sub_menu_accounts_add:
122             return ((ActividadPrincipal) getActivity()).
123                 cambiarFragmento(new FragAgregarEditarCuenta());
124         default:
125             return super.onOptionsItemSelected(item);
126     }
127 }
128
129 //Creación del menu contextual para click prolongado en una
130 Account
131 @Override
132 public void onCreateContextMenu(ContextMenu menu, View v,
133     ContextMenu.ContextMenuInfo menuInfo) {
134     super.onCreateContextMenu(menu, v, menuInfo);
135
136     ExpandableListView.ExpandableListContextMenuInfo info = (
137     ExpandableListView.ExpandableListContextMenuInfo) menuInfo;
138     int type = ExpandableListView.getPackedPositionType(info.
139     packedPosition);
140
141     if (type == ExpandableListView.PACKED_POSITION_TYPE_CHILD) {
142         MenuInflater inflater = getActivity().getMenuInflater();
143         inflater.inflate(R.menu.cont_menu_cuentas, menu);
144     }
145 }

```

```

141
142 //Creación de la funcionalidad del menu contextual
143 @Override
144 public boolean onOptionsItemSelected(MenuItem item) {
145     if (!((ActividadPrincipal) getActivity()).isSesionIniciada())
146     {
147         return false;
148     }
149     ExpandableListView.ExpandableListContextMenuInfo info = (
150     ExpandableListView.ExpandableListContextMenuInfo) item.
151     getMenuInfo();
152     Integer groupIndex = ExpandableListView.
153     getPackedPositionGroup(info.packedPosition);
154     Integer childIndex = ExpandableListView.
155     getPackedPositionChild(info.packedPosition);
156     selectedAccount = adapter.getChild(groupIndex, childIndex);
157
158     switch (item.getItemId()) {
159     case R.id.cont_menu_accounts_edit:
160         FragAgregarEditarCuenta fragAgregarEditarCuenta = new
161         FragAgregarEditarCuenta();
162         Bundle args = new Bundle();
163         args.putString(ColCuenta.NOMBRE.toString(),
164         selectedAccount.getNombre());
165         fragAgregarEditarCuenta.setArguments(args);
166         ((ActividadPrincipal) getActivity()).cambiarFragmento
167         (fragAgregarEditarCuenta);
168         return true;
169
170     case R.id.cont_menu_accounts_delete:
171         AlertDialog alertDialog = new AlertDialog.Builder(
172         getActivity()).create();
173         alertDialog.setTitle("Eliminación de Cuenta");
174         alertDialog.setMessage("¿Está seguro que desea
175         eliminar la cuenta " + selectedAccount.getNombre() +
176         "?");
177         alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE,
178         "NO", new DialogInterface.OnClickListener() {
179             public void onClick(DialogInterface dialog, int
180             which) {
181                 dialog.dismiss();
182             }
183         });
184         alertDialog.setButton(AlertDialog.BUTTON_POSITIVE,
185         "SÍ", new DialogInterface.OnClickListener() {
186             public void onClick(DialogInterface dialog, int
187             which) {
188                 if (cuentaDAO.eliminarUna(selectedAccount.
189                 getNombre()) > 0) {
190                     fillAccountsInfo();
191                     adapter.notifyDataSetChanged();
192                     ((ActividadPrincipal) getActivity()).
193                     actualizarBundles(Cuenta.class,
194                     selectedAccount.getNombre(), null);
195                 }
196             }
197         });
198     }
199 }

```

```

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
195             .seleccionarPorCategoria(categoria.getNombre())) {
196             Cuenta cuenta = cuentaDAO.seleccionarUna(
197                 categoriaCuenta.getNombreCuenta());
198             if (cuenta != null) {
199                 Contrasenna contrasenna = contrasennaDAO.
200                     seleccionarUltimaPorCuenta(cuenta.getNombre());
201                 String fecha = "2000/01/01";
202                 if (contrasenna != null) {
203                     fecha = contrasenna.getFecha();
204                 }
205                 cuentas.add(new CuentaConFecha(cuenta.getNombre(
206                     ), cuenta.getDescripcion(), cuenta.getValidez(),
207                     fecha));
208             } else {
209                 Log.e(TAG, "No se encontró la cuenta con el
210                     nombre: " + categoriaCuenta.getNombreCuenta());
211             }
212         }
213         listaCategorias.add(new CategoriaListaCuentas(categoria.
214             getNombre(), categoria.getPosicion(), cuentas));
215     }
216
217     List<CuentaConFecha> cuentas = new ArrayList<>();
218     for (Cuenta cuenta : cuentaDAO.seleccionarTodas()) {
219         Contrasenna contrasenna = contrasennaDAO.
220             seleccionarUltimaPorCuenta(cuenta.getNombre());
221         String fecha = "2000/01/01";
222         if (contrasenna != null) {
223             fecha = contrasenna.getFecha();
224         }
225         cuentas.add(new CuentaConFecha(cuenta.getNombre(), cuenta
226             .getDescripcion(), cuenta.getValidez(), fecha));
227     }
228     listaCategorias.add(new CategoriaListaCuentas("", null,
229         cuentas));
230 }
231 }

```

```

1  package cl.theroot.passbank.fragmento;
2
3
4  import android.app.AlertDialog;
5  import android.app.Fragment;
6  import android.content.DialogInterface;
7  import android.os.Bundle;
8  import android.view.LayoutInflater;
9  import android.view.Menu;
10 import android.view.MenuInflater;
11 import android.view.MenuItem;
12 import android.view.View;
13 import android.view.ViewGroup;
14 import android.widget.ListView;
15 import android.widget.TextView;
16
17 import java.util.ArrayList;
18 import java.util.List;
19
20 import cl.theroot.passbank.ActividadPrincipal;
21 import cl.theroot.passbank.adaptador.AdapCuentas;
22 import cl.theroot.passbank.R;
23 import cl.theroot.passbank.dominio.Categoria;
24 import cl.theroot.passbank.dominio.CategoriaCuenta;
25 import cl.theroot.passbank.dominio.Cuenta;
26 import cl.theroot.passbank.datos.CategoriaCuentaDAO;
27 import cl.theroot.passbank.datos.CategoriaDAO;
28 import cl.theroot.passbank.datos.CuentaDAO;
29 import cl.theroot.passbank.datos.nombres.ColCategoria;
30
31
32 /**
33  * A simple {@link Fragment} subclass.
34  */
35 public class FragDetalleCategoria extends Fragment {
36     private AdapCuentas adaptador;
37     private List<Cuenta> cuentas = new ArrayList<>();
38     private Bundle bundle;
39     private String nombreCategoria = null;
40
41     private CategoriaDAO categoriaDAO;
42     private CategoriaCuentaDAO categoriaCuentaDAO;
43     private CuentaDAO cuentaDAO;
44
45     public FragDetalleCategoria() {
46         // Required empty public constructor
47     }
48
49
50     @Override
51     public View onCreateView(LayoutInflater inflater, ViewGroup
52     container, Bundle savedInstanceState) {
53         setHasOptionsMenu(true);
54         cuentaDAO = new CuentaDAO(getActivity().getApplicationContext
55         ());

```

```

54     categoriaCuentaDAO = new CategoriaCuentaDAO(getActivity().
55     getApplicationContext());
56     categoriaDAO = new CategoriaDAO(getActivity().
57     getApplicationContext());
58
59     View view = inflater.inflate(R.layout.
60     fragmento_detalle_categoria, null);
61     TextView TV_subTitulo = view.findViewById(R.id.TV_subTitulo);
62     TextView TV_name = view.findViewById(R.id.TV_name);
63     ListView listView = view.findViewById(R.id.LV_cuentas);
64
65     bundle = getArguments();
66     if (bundle != null) {
67         nombreCategoria = bundle.getString(ColCategoria.NOMBRE.
68         toString());
69         if (nombreCategoria != null) {
70             TV_name.setText(nombreCategoria);
71             for (CategoriaCuenta categoriaCuenta :
72             categoriaCuentaDAO.seleccionarPorCategoria(
73             nombreCategoria)) {
74                 Cuenta cuenta = cuentaDAO.seleccionarUna(
75                 categoriaCuenta.getNombreCuenta());
76                 if (cuenta != null) {
77                     cuentas.add(cuenta);
78                 }
79             }
80         }
81
82         adaptador = new AdapCuentas(getActivity(), cuentas,
83         nombreCategoria);
84         listView.setAdapter(adaptador);
85         listView.setSelector(android.R.color.transparent);
86
87         if (cuentas.isEmpty()) {
88             TV_subTitulo.setVisibility(View.INVISIBLE);
89         }
90         getActivity().invalidateOptionsMenu();
91         return view;
92     }
93
94     @Override
95     public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
96     {
97         inflater.inflate(R.menu.sub_menu_detalle_categoria, menu);
98         super.onCreateOptionsMenu(menu, inflater);
99     }
100
101     @Override
102     public void onPrepareOptionsMenu(Menu menu) {
103         if (adaptador != null) {
104             menu.findItem(R.id.
105             sub_menu_detalle_categoria_habilitar_orden).setEnabled(
106             adaptador.getCount() > 1);
107         }
108     }

```

```

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         {
105             return false;
106         }
107         switch (item.getItemId()) {
108             case R.id.sub_menu_detalle_categoria_volver:
109                 getActivity().onBackPressed();
110                 return true;
111             case R.id.sub_menu_detalle_categoria_editar:
112                 FragAgregarEditarCategoria fragAgregarEditarCategoria
113                 = new FragAgregarEditarCategoria();
114                 Bundle args = new Bundle();
115                 args.putString(ColCategoria.NOMBRE.toString(),
116                 nombreCategoria);
117                 fragAgregarEditarCategoria.setArguments(args);
118                 return ((ActividadPrincipal) getActivity()).
119                 cambiarFragmento(fragAgregarEditarCategoria);
120             case R.id.sub_menu_detalle_categoria_eliminar:
121                 AlertDialog alertDialog = new AlertDialog.Builder(
122                 getActivity()).create();
123                 alertDialog.setTitle("Eliminación de una Categoría");
124                 alertDialog.setMessage("¿Está seguro que desea
125                 eliminar la categoría " + nombreCategoria + "? Las
126                 cuentas asociadas no se eliminarán, pero ya no se
127                 relacionarán con esta categoría.");
128                 alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE,
129                 "NO", new DialogInterface.OnClickListener() {
130                     public void onClick(DialogInterface dialog, int
131                     which) {
132                         dialog.dismiss();
133                     }
134                 });
135                 alertDialog.setButton(AlertDialog.BUTTON_POSITIVE,
136                 "SÍ", new DialogInterface.OnClickListener() {
137                     public void onClick(DialogInterface dialog, int
138                     which) {
139                         if (categoriaDAO.eliminarUna(nombreCategoria)
140                         > 0) {
141                             ((ActividadPrincipal) getActivity()).
142                             cambiarFragmento(new FragCategorias());
143                             ((ActividadPrincipal) getActivity()).
144                             actualizarBundles(Categoria.class,
145                             nombreCategoria, null);
146                         }
147                         dialog.dismiss();
148                     }
149                 });
150                 alertDialog.show();
151                 return true;
152             case R.id.sub_menu_detalle_categoria_habilitar_orden:

```

FragDetalleCategoria.java

4/4

```
137
138 
139
140
141
142
143
144
145
146
147
148
149
```

```
        item.setChecked(!item.isChecked());
        if (item.isChecked()) {
            adaptador.setOcultarFlechas(false);
        } else {
            adaptador.setOcultarFlechas(true);
        }
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

```
1 package cl.theroot.passbank.fragmento;
2
3 import android.accounts.AccountManager;
4 import android.app.Fragment;
5 import android.app.ProgressDialog;
6 import android.content.Intent;
7 import android.content.IntentSender;
8 import android.content.pm.ActivityInfo;
9 import android.content.res.Configuration;
10 import android.media.VolumeShaper;
11 import android.os.Bundle;
12 import android.support.annotation.NonNull;
13 import android.support.annotation.Nullable;
14 import android.text.Editable;
15 import android.text.TextWatcher;
16 import android.util.Log;
17 import android.view.LayoutInflater;
18 import android.view.Menu;
19 import android.view.MenuInflater;
20 import android.view.MenuItem;
21 import android.view.View;
22 import android.view.ViewGroup;
23 import android.widget.EditText;
24 import android.widget.ImageView;
25 import android.widget.TextView;
26
27 import com.google.android.gms.auth.GoogleAuthUtil;
28 import com.google.android.gms.common.AccountPicker;
29 import com.google.android.gms.common.ConnectionResult;
30 import com.google.android.gms.common.GoogleApiAvailability;
31 import com.google.android.gms.common.api.GoogleApiClient;
32 import com.google.android.gms.common.api.ResultCallback;
33 import com.google.android.gms.common.api.Status;
34 import com.google.android.gms.drive.Drive;
35 import com.google.android.gms.drive.DriveApi;
36 import com.google.android.gms.drive.DriveContents;
37 import com.google.android.gms.drive.DriveFile;
38 import com.google.android.gms.drive.DriveFolder;
39 import com.google.android.gms.drive.DriveId;
40 import com.google.android.gms.drive.Metadata;
41 import com.google.android.gms.drive.MetadataBuffer;
42 import com.google.android.gms.drive.MetadataChangeSet;
43
44 import java.io.FileInputStream;
45 import java.io.FileNotFoundException;
46 import java.io.IOException;
47 import java.io.InputStream;
48 import java.io.OutputStream;
49 import java.util.ArrayList;
50 import java.util.List;
51
52 import cl.theroot.passbank.ActividadPrincipal;
53 import cl.theroot.passbank.CustomToast;
54 import cl.theroot.passbank.R;
55 import cl.theroot.passbank.dominio.Parametro;
```

```

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

```

```

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().
150 toString();
151 if (email.isEmpty()) {
152     new CustomToast(getActivity().
153     getApplicationContext(), "¡Acción
154     Cancelada!\nDebe seleccionar una cuenta antes de
155     realizar el respaldo.");
156     return true;
157 }
158
159 if (mGoogleApiClient != null && mGoogleApiClient.
160 isConnected()) {
161     new CustomToast(getActivity().
162     getApplicationContext(), "¡Acción Cancelada!\nYa
163     se está ejecutando la creación del respaldo de
164     los datos.");
165     return true;
166 }
167
168 //Deshabilitar rotación de pantalla...
169 if (getActivity().getResources().getConfiguration().
170 orientation == Configuration.ORIENTATION_LANDSCAPE) {
171     getActivity().setRequestedOrientation(
172     ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
173 } else {
174     getActivity().setRequestedOrientation(
175     ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
176 }
177
178 parametroDAO.actualizarUna(new Parametro(
179 NombreParametro.CUENTA_GOOGLE.toString(), email, null
180 ));
181 mensajeProgreso.show();
182 mGoogleApiClient = new GoogleApiClient.Builder(
183 getActivity())
184     .addApi(Drive.API)
185     .addScope(Drive.SCOPE_APPFOLDER)
186     .addConnectionCallbacks(this)
187     .addOnConnectionFailedListener(this)
188     .setAccountName(email)
189     .build();
190 mGoogleApiClient.connect();
191
192 return true;
193 default:
194     return super.onOptionsItemSelected(item);
195 }
196 }
197
198 @Override
199 public void onClick(View view) {
200     switch (view.getId()) {
201     case R.id.ET_cuenta_seleccionada:

```

```

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

```

```

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.
257                 BANCO_CONTRASENNAS_RESPALDO.
258                 toString())
259         .setMimeType(
260                 "application/x-sqlite3")
261         .build();
262
263         //Se copian los datos de la base de
264         //datos local en el contenido para el
265         //nuevo respaldo
266         DriveContents contents =
267         driveContentsResult.getDriveContents
268         ();
269         InputStream in = null;
270         OutputStream out = null;
271         try {
272             out = contents.getOutputStream();
273             in = new FileInputStream(dbFile);
274
275             byte[] buffer = new byte[1024];
276             int read;
277             while ((read = in.read(buffer))
278                 != -1) {
279                 out.write(buffer, 0, read);
280             }
281         } catch(FileNotFoundException ex) {
282             Log.e(TAG, "No se encontró el
283             archivo local de la base de
284             datos", ex);
285             terminarConexion(";Error!\nNo se
286             pudo realizar su respaldo,
287             intente más tarde.");
288             return;
289         } catch(IOException ex) {
290             Log.e(TAG, "No se pudo hacer
291             lectura de la base de datos", ex);
292             terminarConexion(";Error!\nNo se
293             pudo realizar su respaldo,
294             intente más tarde.");
295             return;
296         } finally {
297             if (in != null) {
298                 try {
299                     in.close();
300                 } catch (IOException ex) {
301                     //Nada
302                 }
303             }
304             if (out != null) {
305                 try {
306                     out.close();
307                 } catch (IOException ex) {
308                     //Nada
309                 }
310             }
311         }

```

```

296     }
297
298     //Se crea el nuevo respaldo, con los
299     //metadatos y contenidos obtenidos
300     //anteriormente
301     appFolder.createFile(mGoogleApiClient
302     , metadataChangeSet, contents).
303     setResultCallback(new ResultCallback<
304     DriveFolder.DriveFileResult>() {
305         @Override
306         public void onResult(@NonNull
307         DriveFolder.DriveFileResult
308         driveFileResult) {
309             if (!driveFileResult.
310             getStatus().isSuccess()) {
311                 Log.e(TAG, "Error al
312                 crear el archivo de
313                 respaldo en Google Drive"
314                 );
315                 terminarConexion(
316                 "¡Error!\nNo se pudo
317                 crear el archivo de
318                 respaldo en Google
319                 Drive, intente más
320                 tarde.");
321                 return;
322             }
323             Log.i(TAG, "El archivo de
324             respaldo fue creado
325             exitosamente con ID: " +
326             driveFileResult.getDriveFile
327             ().getDriveId().toString());
328
329             //Se comienza la eliminación
330             //de los respaldos anteriores
331             //al recién creado
332             if (!listaIds.isEmpty()) {
333                 final List<DriveId>
334                 idsProcesados = new
335                 ArrayList<>();
336                 for (final DriveId
337                 driveId : listaIds) {
338                     Log.i(TAG,
339                     "Eliminando el
340                     archivo con ID: " +
341                     driveId.toString());
342                     DriveFile driveFile =
343                     driveId.asDriveFile
344                     ();
345                     driveFile.delete(
346                     mGoogleApiClient).
347                     setResultCallback(new
348                     ResultCallback<
349                     Status>() {
350                         @Override

```

```

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

```

```
352 Log.e(TAG, "La conexión GoogleApiClient ha fallado: " +
353       connectionResult.toString());
354 if (!connectionResult.hasResolution()) {
355     Log.e(TAG, "No existe una solución al problema... Cancelando
356     exportación...");
357     terminarConexion(null);
358     GoogleApiAvailability.getInstance().getErrorDialog(
359     getActivity(), connectionResult.getErrorCode(), 0).show();
360     return;
361 }
362 try {
363     Log.i(TAG, "Tratando de resolver el problema...");
364     connectionResult.startResolutionForResult(getActivity(),
365     SOLUCIONADOR_PROBLEMAS);
366 } catch (IntentSender.SendIntentException ex) {
367     Log.e(TAG, "Excepción mientras se trataba de iniciar la
368     actividad de resolución de problemas", ex);
369 }
370 }
371
372 private void terminarConexion(String mensaje) {
373     mGoogleApiClient.disconnect();
374     mGoogleApiClient = null;
375     mensajeProgreso.dismiss();
376     if (mensaje != null) {
377         new CustomToast(getActivity().getApplicationContext(),
378         mensaje);
379     }
380     //Habilitar rotación de pantalla...
381     getActivity().setRequestedOrientation(ActivityInfo.
382     SCREEN_ORIENTATION_UNSPECIFIED);
383 }
384 }
```

```

1  package cl.theroot.passbank.fragmento;
2
3  import android.app.AlertDialog;
4  import android.content.DialogInterface;
5  import android.os.Bundle;
6  import android.app.Fragment;
7  import android.view.LayoutInflater;
8  import android.view.View;
9  import android.view.ViewGroup;
10 import android.widget.Button;
11 import android.widget.EditText;
12
13 import cl.theroot.passbank.ActividadPrincipal;
14 import cl.theroot.passbank.Cifrador;
15 import cl.theroot.passbank.Desfragmentador;
16 import cl.theroot.passbank.ExcepcionBancoContrasennas;
17 import cl.theroot.passbank.R;
18 import cl.theroot.passbank.dominio.Parametro;
19 import cl.theroot.passbank.datos.CategoriaCuentaDAO;
20 import cl.theroot.passbank.datos.CategoriaDAO;
21 import cl.theroot.passbank.datos.ContrasennaDAO;
22 import cl.theroot.passbank.datos.CuentaDAO;
23 import cl.theroot.passbank.datos.PalabraDAO;
24 import cl.theroot.passbank.datos.ParametroDAO;
25 import cl.theroot.passbank.datos.nombres.NombreParametro;
26
27 public class FragInicioSesion extends Fragment implements View.
   OnClickListener{
28     private static final String TAG = "BdC-FragInicioSesion";
29
30     private EditText ET_contrasenna;
31
32     private ParametroDAO parametroDAO;
33
34     @Override
35     public View onCreateView(LayoutInflater inflater, ViewGroup container
   , Bundle savedInstanceState) {
36         View view = inflater.inflate(R.layout.fragmento_inicio_sesion,
   null);
37         Button B_login = view.findViewById(R.id.B_login);
38         B_login.setOnClickListener(this);
39         ET_contrasenna = view.findViewById(R.id.ET_password);
40         getActivity().invalidateOptionsMenu();
41
42         parametroDAO = new ParametroDAO(getActivity().
   getApplicationContext());
43
44         //Tratar de crear la base de datos para el diccionario de palabras
45         new PalabraDAO(getActivity().getApplicationContext());
46         //Tratar de desfragmentar la base de datos de la aplicación
47         Desfragmentador.intentarDesfragmentación(getActivity().
   getApplicationContext());
48
49         //Imprimir datos de la base de datos para debuggeo

```

```

50     new CategoriaDAO(getActivity().getApplicationContext()).imprimir
51     ();
52     new CuentaDAO(getActivity().getApplicationContext()).imprimir();
53     new CategoriaCuentaDAO(getActivity().getApplicationContext()).
54     imprimir();
55     new ContrasennaDAO(getActivity().getApplicationContext()).
56     imprimir();
57     new ParametroDAO(getActivity().getApplicationContext()).imprimir
58     ();
59     return view;
60 }
61
62 @Override
63 public void onClick(View v) {
64     try {
65         switch (v.getId()) {
66             case R.id.B_login:
67                 String contrasenna = ET_contrasenna.getText().
68                 toString();
69                 if (contrasenna.isEmpty()) {
70                     throw new ExcepcionBancoContrasennas("Error -
71                     Llave Maestra Requerida", "Se requiere el
72                     ingreso de su Llave Maestra para el inicio de
73                     sesión.");
74                 }
75                 Parametro parSalt = parametroDAO.seleccionarUno(
76                 NombreParametro.SAL_HASH.toString());
77                 Parametro parHash = parametroDAO.seleccionarUno(
78                 NombreParametro.RESULTADO_HASH.toString());
79                 String[] saltyHashObt = Cifrador.genHashedPass(
80                 contrasenna, parSalt.getValor());
81                 if (!saltyHashObt[1].equals(parHash.getValor())) {
82                     throw new ExcepcionBancoContrasennas("Error -
83                     Llave Maestra Incorrecta", "La Llave Maestra
84                     ingresada no es correcta, favor intentar de
85                     nuevo.");
86                 }
87                 Parametro parSaltEncr = parametroDAO.seleccionarUno(
88                 NombreParametro.SAL_ENCRYPTACION.toString());
89                 String[] saltyHashEncr = Cifrador.genHashedPass(
90                 contrasenna, parSaltEncr.getValor());
91                 ((ActividadPrincipal) getActivity()).userLogIn(
92                 saltyHashEncr[1]);
93                 break;
94             }
95         } catch (ExcepcionBancoContrasennas ex) {
96             AlertDialog alertDialog = new AlertDialog.Builder(getActivity
97             ()).create();
98             alertDialog.setTitle(ex.getTitulo());
99             alertDialog.setMessage(ex.getMensaje());
100            alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK", new
101            DialogInterface.OnClickListener() {
102                public void onClick(DialogInterface dialog, int which) {
103                    dialog.dismiss();
104                }
105            });
106            alertDialog.show();
107        }
108    }
109 }

```