UNIVERSIDAD DE GRANADA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA Y TELECOMUNICACIONES

Programación de Dispositovos Móviles

MEMORIA TÉCNICA PARA LA APLICACIÓN

SAFEPASSWORD

Jose Manuel Osuna Luque

Índice

1. Introducción

En este proyecto se ha construido una aplicación Android para la gestión de las contraseñas. De este modo, un usuario, previo registro de una contraseña, se le creará una cuenta en la que podrá almacenar todas sus contraseñas, para que, como es muy normal, la persona no las olvide y tenga un lugar al que poder recurrir en caso de no acordarse.

Su funcionamiento es sencillo. Una vez el usuario de la aplicación ha registrado una contraseña, puede iniciar sesión. Al hacerlo, aparece un listado con las contraseñas que haya guardado (sin un límite). En general, en la aplicación se puede hacer lo siguiente:

- Añadir una contraseña. Se le deberá añadir obligatoriamente un nombre para identificarla, un usuario correspondiente una contraseña (que no sea menor que 4 caracteres). Adiccionalmente, se podrá añadir la URL de la página a la que pertenezca. También se puede elegir una categoría y un icono representativo.
- Visualizar una contraseña al pulsar alguna ya registrada. Puede verse todo el contenido de esta, salvo la contraseña, para lo cual será necesario introducir la contraseña del usuario de la aplicación o el pin registrado previamente.
- Se podrá editar una contraseña ya existente deslizando hacia la izquierda dicha contraseña.
- Como caso opuesto (deslizando hacia la derecha), el usuario podrá borrar la contraseña.
- Se puede hacer un filtro para mostrar solo contraseñas pertenecientes, por ejemplo, a la categoría Red Social".
- Si la contraseña tiene una dirección web asociada, es posible abrir este enlace en una pestaña del navegador por defecto del dispositivo movil.
- Modificar el pin ya existente o registrar uno nuevo si aun no estaba registrado. El pin se usa como una verificación de usuario rápida de cuatro dígitos, en vez de introducir la contraseña siempre que se quiera ver una contraseña.
- Se podrán hacer copias de seguridad a nivel local, por si el usuario quisiera recuperarla en caso de haber perdido o eliminado por error una contraseña de la que no quería deshacerse.
- Eliminar el Usuario, lo que implicaría la perdida de todos los datos relacionados con esa contraseña que el usuario usó para registrarse. Se eliminarían todas las contraseñas, el pin que el usuario pudiera haber gestionado, copias de seguridad o correos vinculados.

2. Desarrollo

Figura 1: SafePassword

Figura 2: Icono

SafePassword, una aplicación de gestión de contraseñas seguras. En este apartado se hablará de cómo se ha construido la aplicación, empezando por lo más esencial, como es la elección de unos colores corporativos y una identificación (como puede ser el logo), pasando por todas las clases que componen la aplicación Android y detallando en profundidad los aspectos más llamativos e importantes que han surgido a la hora de la creación de la misma.

2.1. Esquema de Color

Se ha optado por una gama de colores de tonalidades azul con una gama de colores alternativa y adyacente al color principal elegido. Para obtener estos colores se ha utilizado la herramienta de Paletton.

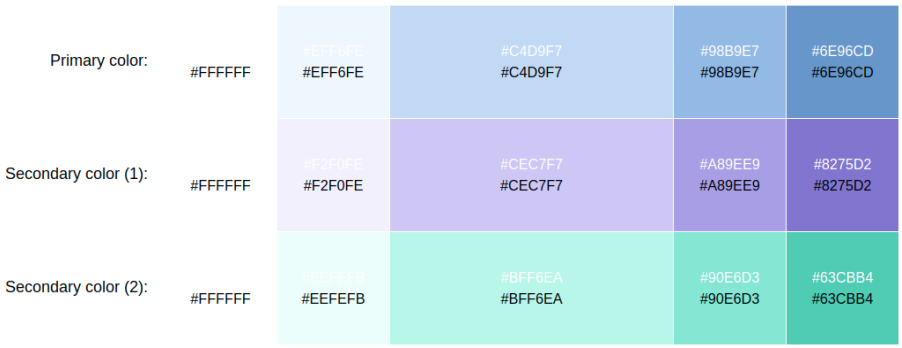

Figura 3: Esquema de Colores

2.2. Planteamiento

Para el desarrollo de la Aplicación se ha elegido el IDE de Google, Android Studio. Además, para la contrucción de la aplicación, se ha elegido el SDK 30 de Google, la última versión de Android, aunque se ha fijado una API mínima donde la aplicación funcionaría, la API v27. Esto quiere decir que la aplicación desarrollada funcionaría sin problemas en dispositivos de versión 8.1 de Android en adelante.

El por qué de esta decisión se debe a que más del cincuenta por ciento de los dispositivos ya usa dicha versión. Además, trabajar con versiones inferiores a esta limitaría el uso de nuevas tecnologías implementadas en Android para las versiones más recientes.

El lenguaje de programación usado para la aplicación ha sido JAVA, que si bien es conocido que Kotlin es la nueva apuesta para Android, es aun un lenguaje en auge del que hay menos información en Internet para trabajar cómodamente con él (aunque no se niega un re-desarrollo en dicho lenguaje). A esto se añade, como ya es conocido por el funcionamiento de Android, que se han usado ficheros XML para aplicar diseños a las distintas actividades y navegación posible en la aplicación.

Por último, decir que se ha optado por un lanzamiento bilingüe de la aplicación, de forma que reconozca el idioma definido en el dispositivo para que esta se muestre en Inglés o Español.

Figura 4: JAVA

Figura 5: Android Studio

2.3. Navegacion de la Aplicación

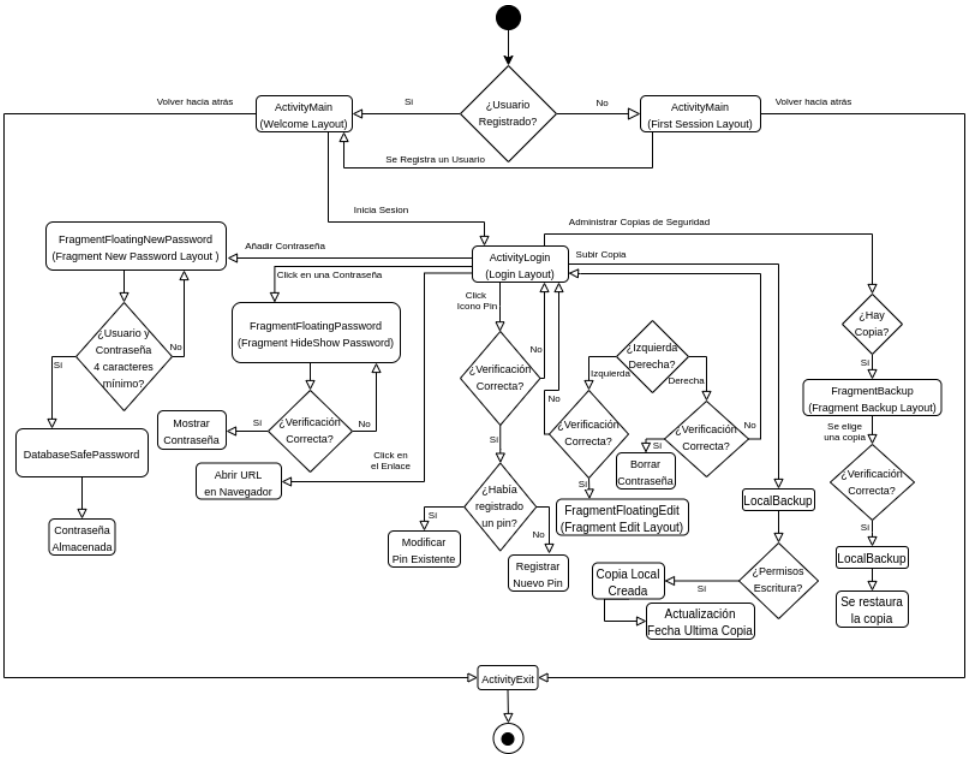

Figura 6: Diagrama de Navegacion

En la imagen anterior se puede apreciar un Diagrama de Navegación de la aplicación, detallando a grandes rasgos el recorrido que hace la aplicación desarrollada según la acción del usuario. A continuación, se van a nombrar las Actividades que componen el proyecto y se hablará más en profundidad sobre el código y la programación llevada a cabo, es decir, la implementación de la misma.

2.4. Estructura

La aplicación se resuelve en torno a una base de datos local controlada por la clase *DatabaseSafePassword*. Esta clase estática gestiona todas las acciones hechas en la base de datos con un método por cada Sentencia SQL aplicada. La base de datos es la típica con la que se trabaja con Android en modo local, SQLite.

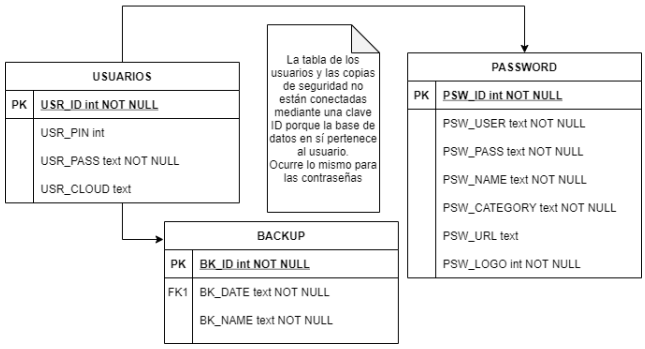

Figura 7: Estructura de la Base de Datos

Además, para cada elemento de la base de datos se ha creado una clase. Es decir, se tiene una clase para gestionar el usuario registrado, *UserDatabase*, otra clase para recoger cada contraseña, *PasswordDatabase*, y una última clase para controlar las copias de seguridad, *BackupUserDatabase*.

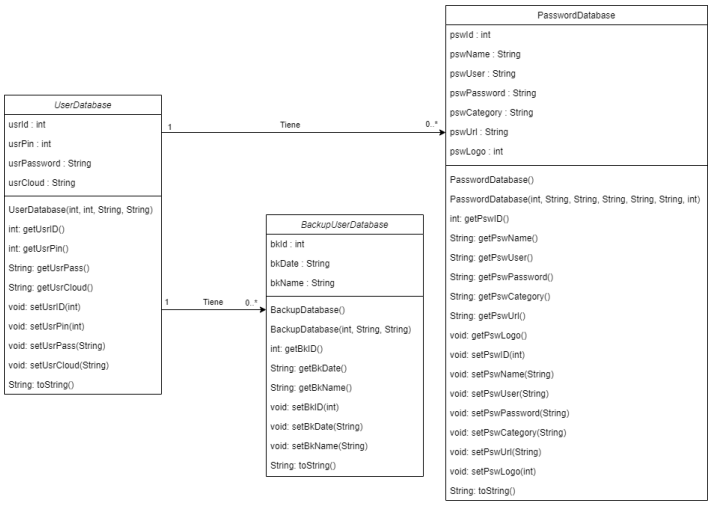

Figura 8: Elementos Principales

En la clase de la base de datos se recogen los métodos necesarios para poder gestionar la aplicación, el tema de las contraseñas que pertenecen a cada usuario, y las copias de seguridad, así como al propio usuario, que en este caso solo está compuesto de una contraseña.

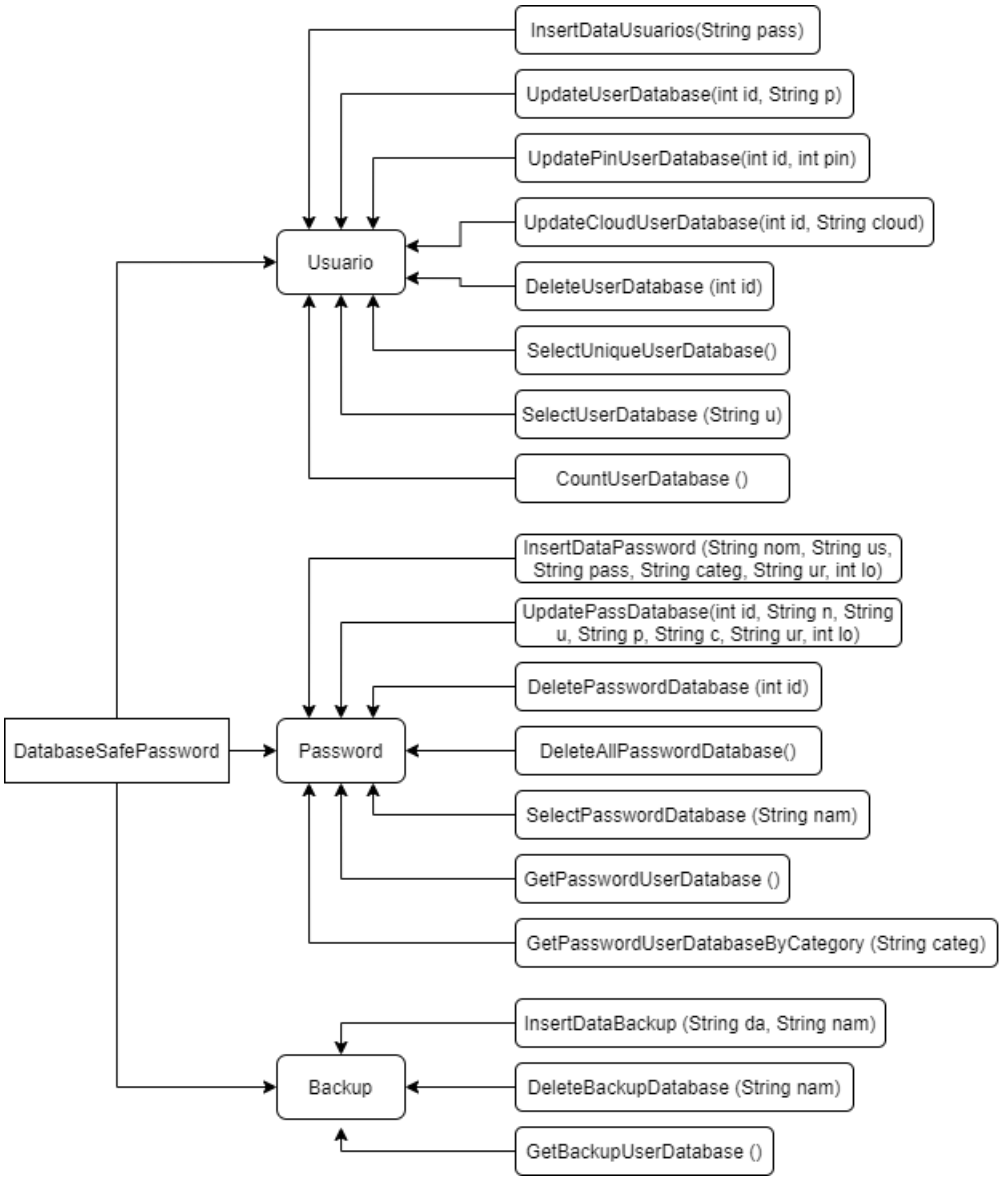

Figura 9: Gestión Base de Datos

3. Implementación

3.1. ActivityMain

Esta es la primera actividad que se crea al iniciar la aplicación. Esta se característica por contar con dos Layout (apariencias, vistas) que se visualizan según lo que la aplicación detecte respecto a si hay datos guardados o no con anterioridad. En caso de haberlos, mostrará el diseño de Inicio de Sesión, pero, si por el contrario no detecta una base de datos creada en la que haya un usuario registrado, entonces mostrará un diseño donde lo único que puede hacerse es registrarse (introduciendo una contraseña).

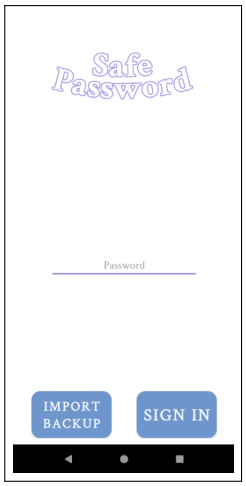

Figura 10: No Hay Usuario Registrado

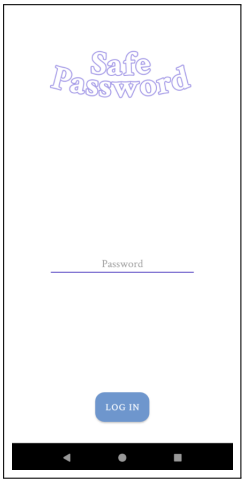

Figura 11: Usuario Registrado

Para conseguir esto, es suficiente con hacer una comprobación con el objeto que se crea de la base de datos, y comprobar si existe algún usuario registrado. Si el resultado devuelve *TRUE*, quiere decir que existe al menos un usuario y entonces se carga la apareciencia en la que no aparece un botón de registro mediante *setContentView).*

```
if (database.CountUserDatabase()){
    setContentView(R.layout.activity_welcome);
    ...
}else{
    setContentView(R.layout.activity_first_session);
    ...
}
```

3.2. ActivityLogin

Dicha Activity es el eje central de la aplicación. Es la que controla todo lo que el usuario puede hacer, desde poder ver en una cuadrícula sus contraseñas almacenadas, eliminarlas o editarlas, hasta borrar el usuario en cuestión, cuya consecuencia es la pérdida absoluta e irreparable de todas las contraseñas almacenadas.

Figura 12: Activity Login

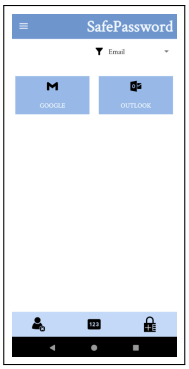

Figura 13: Filtro Email

Para crear esta vista, se ha usado un RecyclerView, combinado con una estructura GridLayout. Para la gestión automática de las contraseñas, al recyclerView se le ha creado una clase Adapter (*AdapterPassword*) que controla que cada objeto de tipo PasswordDatabase (que es la clase creada para crear objetos de tipo contraseña). En la clase AdapterPassword el método *PasswordViewHolder* es donde se cargan y se identifican los elementos de la vista que se usará para representar la contraseña en la vista principal.

```
public PasswordViewHolder(View itemPassView){
    super(itemPassView);
    tv_item_password_name = itemPassView.findViewById(R.id.it_name);
    iv_item_password_logo = itemPassView.findViewById(R.id.it_logo);
}
```

El metodo *onCreateViewHolder* se usa dentro del adapter para definir el layout que tomará para mostrar la contraseña en el *RecyclerView* de la vista principal. Cada vez que se recoge una contraseña se le aplicará, mediante el adapter el layout, y se cargarán los datos de forma que cada contraseña se representa con dicha vista dentro del RecyclerView. Por último, pero no menos importante, en este método, para que al pulsar sobre alguna de las vistas generadas representando a cada contraseña se le pueda dar alguna funcionalidad, se deja indicado el *setOnClickListener* de esta forma, para que en la clase *ActivityLogin* se defina al crear el Adapter.

```
@Override
public AdapterPassword.PasswordViewHolder
    onCreateViewHolder(ViewGroup parent, int viewType){

    View itemView = LayoutInflater.from(parent.getContext()).
        inflate(R.layout.item_password, parent, false);

    PasswordViewHolder contenedor =
        new PasswordViewHolder(itemView);

    itemView.setOnClickListener(this);

    return contenedor;
}
```

Para que queda vista se carga con el nombre y el logo de cada contraseña, para entender, se explica como que al crear el Adapter, se recogen anteriormente todas las contraseñas que se van a visualizar en el GridLayout, después de eso, todas se posicionan en un lugar dentro del Grid. Hasta aquí, podría decirse que se tiene un conjunto de contraseñas repartidas por el GridLayout pero sin visualización ninguna. El metodo siguiente, junto a los anteriores que recogen el Layout personalizado para el Item o la recogida e identificación de los elementos, hace que se aplique la vista y además se escriban sus datos correctamente.

```
@Override
public void onBindViewHolder(PasswordViewHolder viewHolder, int pos) {
    PasswordDatabase item = passwordDataSet.get(pos);

    viewHolder.tv_item_password_name.setText(item.getPswName());

    viewHolder.iv_item_password_logo.setImageResource(item.getPswLogo());
}
```

En la parte de abajo se pueden apreciar tres botones. Comenzando por la izquierda, el primer icono que se aprecia sirve para eliminar todas las contraseñas del usuario, y además al propio usuario. Esto quiere decir que la próxima vez que se entre en la aplicación aparecerá la interfaz de registro. Aunque haya un pin registrado, para poder realizar esta acción es necesario introducir la contraseña.

El botón central sirve para registrar un nuevo PIN en la aplicación, o, en caso de existir ya uno, servirá para modificarlo. El PIN sirve como una verificación más rápida para el usuario, para usarla en vez de introducir la contraseña cuando la aplicación pida verificación para realizar ciertas acciones.

El último botón con forma de candado sirve para añadir una nueva contraseña.

Al pulsar sobre una contraseña existente se abrirá una nueva vista en la que se mostrará la información referente sobre la contraseña, salvo la contraseña, que para poder visualizarse será necesario que el usuario se verifique.

En la parte superior de la pantalla aparece una lista en la que aparecen varias opciones. Al pulsar alguna de las disponibles, las contraseñas se filtrarán, apareciendo solo las de la categoría elegida. Para crear este filtro, simplemente, en la clase del *ActivityLogin* y en la clase de la Base de datos, se ha creado un método alternativo para que al recoger las contraseñas y se pasen al Adapter para crear la vista antes explicada, en esta ocasión solo se devuelvan las contraseñas que superen el filtro.

Por tanto, a esta barra de filtro, que ha sido creada mediante la clase Spinner propia de Android, se le ha modificado su método *onItemSelected(...)* para que actue a razón de la opción elegida en su menú y actualice y cargue las contraseñas filtradas.

```
@Override
public void onItemSelected(AdapterView<?> parent, View view, int position, long id){
    // Se puede recoger el elemento pulsado de la siguiente forma
    String categ = parent.getItemAtPosition(position).toString();

    if(categ.equals("All") || categ.equals("Todo")){
        loadPassword(); //Las devuelve todas
    }else{
        loadPasswordByCategory(categ); //Las devuelve filtradas
    }
}
```

El método llama es una clase privada asíncronada creada para no sobrecargar el hilo principal y hacer las operaciones de base de datos en segundo plano.

```
private void loadPasswordByCategory(String categ){
    new PasswordLoadByCategoryTask().execute(categ);
}
```

La clase privada asíncronada creada está dividida en dos operaciones. La primera, *doInBackground*, como su nombre indica, realiza las operaciones en la base de datos en segundo plano. La segunda, *onPostExecute* se ejecuta cuando la primera pare de la clase termina y obtiene un resultado que pasa a dicho método. En esta segunda parte lo que se hace es, en cierto modo, sobreescribir el adapter y su contenido que ya tuviese, cargándolo con las contraseñas filtradas. Se vuelven a definir tanto el layout usado, como el método de acción al pulsar sobre alguna contraseña, y, una vez el adapter tiene todo el contenido querido, se aplica al *RecyclerView*.

```
private class PasswordLoadByCategoryTask extends AsyncTask<String,
                                        Void, ArrayList<PasswordDatabase>> {
    @Override
    protected ArrayList<PasswordDatabase> doInBackground(String... strings) {
        allPassword =
            ActivityMain.database.GetPasswordUserDatabaseByCategory(strings[0]);

        return allPassword;
    }
    @Override
    protected void onPostExecute(ArrayList<PasswordDatabase> lista) {
        if (lista != null && lista.size() > 0) {
            adapter = new AdapterPassword(lista);
            adapter.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    //Se recoge el objeto de la posicion en la que se hizo click
                    PasswordDatabase pd =
                        lista.get(rviewListPass.getChildAdapterPosition(v));

                    // Crea un intent cargado con los datos de la password
                    Intent showPass = new Intent (ActivityLogin.this,
                                            FragmentFloatingPassword.class);

                    // Se le aplica una animacion
                    ActivityOptions animacion =
                        ActivityOptions.makeCustomAnimation(getApplicationContext(),
                        R.anim.fade_in, R.anim.fade_out);
                    //Se proporciona un contenido extra al Intent
                    showPass.putExtra("ID_PASS", pd);
                    //Se inicializa
                    startActivity(showPass, animacion.toBundle());
                }
            });
            // Se le aplica la funcionalidad del ItemTouchHelper
            ItemTouchHelper itemTouchHelper =
                new ItemTouchHelper(itemTouchHelperCallback);

            itemTouchHelper.attachToRecyclerView(rviewListPass);

            rviewListPass.setAdapter(adapter);
            rviewListPass.setLayoutManager(rviewGridLayoutManager);
            rviewListPass.setPadding(0,50,0,50);
        }
    }
}
```

En el código anterior se puede apreciar en la zona de la parte de *onClick* el objeto de tipo *Intent*. Este objeto es quien permite la navegación entre actividades. Su funcionamiento es sencillo, simplemenete hay que indicar la Actividad actual en la que se está, y a la que se quiere cambiar. Si además, hiciese falta pasar algún tipo de valor o dato, se usaría *putExtra(TAG, VALOR)*.

Por último queda mencionar el menú desplazable que está situado a la izquierda de la Actividad. Este menú cuenta con las opciones referentes a las Copias de Seguridad que el usuario ha hecho, como cual fue la última copia en local, o incluso si este ha iniciado sesión en Google para realizar la sincronización de sus contraseñas en la nube (esto último no está implementando en esta versión).

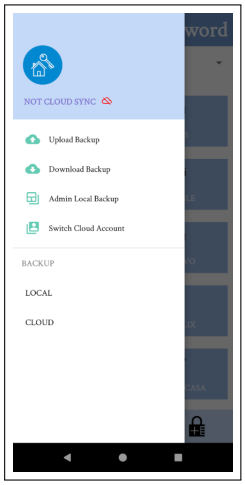

Figura 14: Menu Desplazable - Activity Login

Para crear este menú, se ha hecho uso de la clase DrawerLayout. En la vista usada (Layout) es necesario hacer que sea la etiqueta *DrawerLayout* la que incluya todo el contenido de dicho layout, y además añadir un *NavigationView* ya que está será la vista que se desplaza a la izquierda (es decir, el propio menú que saldrá al deslizar o pulsar en el botón de la izquierda).

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
        ...
        <!-- Este LinearLayout presenta el contenido de la pantalla -->
        <LinearLayout
                ...
        </LinearLayout>

        <!-- Es la vista que se desplaza desde la izquierda -->
        <!-- Para ello es necesario indicar el layout_gravity en START -->
        <com.google.android.material.navigation.NavigationView
                ... />
</androidx.drawerlayout.widget.DrawerLayout>
```

Como este *DrawerLayout* junto al *NavigationView* hacen de menú lateral, sustituyendo el típico menú de "los tres puntitos.ªrriba a la derecha de la *Toolbar*, es necesario definir un menú en el apartado *menu* del proyecto para asignarlo a este *NavigationView*. Esta asignación se hace en el fichero *XML* en la parte de la definición del *NavigationView* en el atributo *app:menu*.

```
<com.google.android.material.navigation.NavigationView
        android:id="@+id/navigation_settings"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:background="@color/white"
        app:headerLayout="@layout/menu_header"
        app:itemIconTint="@color/color_alternative_secondary_dark"
        app:menu="@menu/menu_settings" />
```

Además, para que todo esto se visualiza en la parte de la barra de herramientas, es necesario también definir dicho elemento. Es necesario crear también incluirla en el Layout del *ActivityLogin*.

```
    <!-- La ActionBar se mostrara en el top de la aplicacion -->
    <include
        layout="@layout/custom_toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
```

```
    private Toolbar toolbar;
    // Aplica la Toolbar que reemplazara la ActionBar por defecto
    toolbar = findViewById(R.id.main_toolbar);
    // Se activa la Toolbar en la Aplicacion, pero la personalizada
    setSupportActionBar(toolbar);
```

Para gestionar lo que hace cada elemento del menú, se controla al sobreescribir el método *onNavigationItemSelected(...)*. Este método recibe un objeto de la clase MenuItem del que se puede extraer un identificador único para reconocer qué elemento del menú fue pulsado *item.getItemId()*.

Justo debajo del icono de la aplicación aparecerá *Not Sync Cloud* si el usuario no ha sincronizado la aplicación con un correo de Google; o, en su defecto, si el usuario ha iniciado sesión con una cuenta de Google, aparecerá el nombre del correo.

Al pulsar *Upload Backup* el usuario podrá crear una copia de seguridad (solo local). Al crearse el campo del Menú *LOCAL* se actualiza su nombre con la fecha de creación de esta.

La opción *Admin Local Backup* sirve comprobar cuántas copias de seguridad tiene el usuario.

En *Switch Cloud Account* podrá iniciar sesión en Google, o en el caso de que ya estuviese iniciada la sesión, el usuario podrá cambiar su correo. La versión de la aplicación solo permite hacer un inicio de sesión en Google, pero por problemas dependientes de la actualización de la *API* de *DRIVE* por parte de Google, no es posible acceder y dar permisos de lectura y escritura en la nube personal del usuario de DRIVE, salvo que la aplicación esté subida al Play Store y pase por una serie de protocolos de seguridad por parte de Google. Sin embargo, a continuación se pone un fragmento de código que es suficiente para que la aplicación sea capaz de iniciar sesión en una cuenta Google.

```
// Inicia una actividad para el inicio de sesion en el Servicio Google
private void requestSignIn() {

    GoogleSignInOptions signInOptions =
        new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestEmail()
            .build();

    GoogleSignInClient client = GoogleSignIn.getClient(this, signInOptions);

    // The result of the sign-in Intent is handled in onActivityResult.
    startActivityForResult(client.getSignInIntent(), REQUEST_CODE_SIGN_IN);
}
```

Este fragmento de código iniciará una actividad interna, del propio Android Google que pedirá un correo para iniciar sesión. Si bien en el dispositivo ya hay un correo Google registrado dará la opción de elegirlo, pero también puede iniciarse sesión por vez primera con otro.

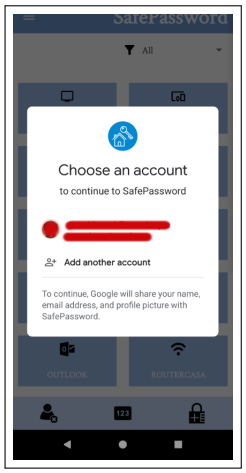

Figura 15: Iniciar Sesión Google

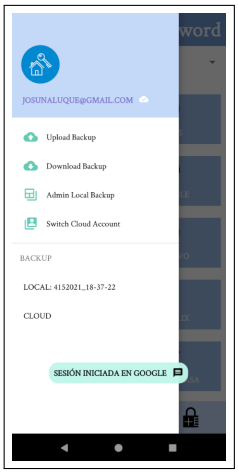

Figura 16: Inicio Google

3.3. FragmentFloatingNewPassword

En este *FragmentActivity* el usuario de la aplicación será capaz de crear una nueva contraseña. Esta vista se activa al pulsar sobre el icono del candado de la Actividad anteriormente explicada.

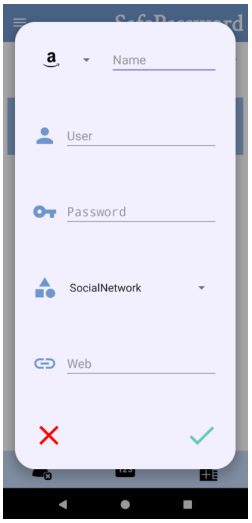

Figura 17: Fragment Activity - Nueva Contraseña

En el primer apartado, el usuario podrá elegir un icono para identificar visualmente el tipo de contraseña que ha guardado. Por ejemplo, puede usar el icono de Instagram, Facebook, Gmail, Outlook. Se han añadido las plataformas y redes sociales más usadas. Si el usuario no encuentra un icono acorde al tipo de contraseña, se han añadido varios iconos visualmente neutros. En esta misma línea, el usuario deberá elegir un nombre para reconocer su contraseña, como puede TWitter o CorreoUGR, algo con lo que luego sea capaz de saber de qué se trataba.

En la siguiente línea, se escribirá el usuario. También este como la contraseña, como pueden tener menos de 4 caracteres. Esto se ha hecho así para evitar campos vacíos tan importantes como estos. Siguiendo con la vista, lo siguiente que se encuentra es un menú desplegable, donde el usuario puede elegir una categoría de entre las posibles opciones. Esto servirá para poder realizar el filtro antes mencionado. Por último, de así quererlo, el usuario puede añadir una URL, un enlace web que defina el sitio donde esta contraseña se usará.

Cuando todos los campos se han rellenado, se realizan las comprobaciones obligatorias para poder guardar la contraseña en la base de datos. Si todo está en orden, se hace una llamada a la Base de Datos y se procede a ejecutar la sentencia necesaria, tras pasar todos los datos necesarios para guardar esta nueva contraseña en la base de datos.

```
//De esta forma se recogen los datos introducidos en los campos

String newName = password_new_name.getText().toString();
String newUser = password_new_user.getText().toString();
String newPass = password_new_pass.getText().toString();
String newCateg = password_new_categ.getSelectedItem().toString();
String newUrl = password_new_url.getText().toString();

if(newPass.length() >= ActivityMain.MINIMUM_LENGTH_CHARACTERS){
    //Llamada a la base de datos para comprobar si la encuentra por el nombre
    PasswordDatabase exist = ActivityMain.database.SelectPasswordDatabase(newName);

    // A la hora de insertar una nueva, esta no puede existir de antes
    // La busqueda hecha no puede devolver una ya existente
    if (exist.getPswID() == -500) {
        //Todo correcto, se inserta en la base de datos
        boolean correcto = ActivityMain.database.InsertDataPassword(newName,
                                    newUser, newPass, newCateg, newUrl, newLogo);

        if (correcto) {
            mensaje.
                toastAdvice(getString(R.string.mensaje_password_new_password_added));
        } else {
            mensaje.
                toastAdvice(getString(R.string.mensaje_generico_error_safepassword));
        }

        finish();

    } else {
        // En cualquier otro caso, el nombre asociado es incorrecto
        // Ya que esta siendo usado en otra
        mensaje.toastAdvice(getString(R.string.mensaje_password_name_incorrecto));
    }

}else{
    mensaje.toastAdvice(getString(R.string.mensaje_password_minimo_caracteres));
}
```

```
// Sentencia SQL para insertar datos en la tabla PASSWORD
INSERT_PASS = "INSERT INTO "+ TABLE_PASSWORD+" ("
    +KEY_PASSWORD_ID+", "+KEY_PASSWORD_NAME+", "+KEY_PASSWORD_USER+", "
    +KEY_PASSWORD_PASS+", "+KEY_PASSWORD_CATEGORY+", "+KEY_PASSWORD_URL+", "
    +KEY_PASSWORD_LOGO+") " +
        "VALUES (null, '"+nom+"', '"+us+"', '"+pass+"', '"
                                            +categ+"', '"+ur+"', '"+lo+"');";

objectDB = getWritableDatabase();

if(objectDB != null){
    objectDB.execSQL(INSERT_PASS);
...
```

A simple vista puede apreciarse que el estilo de dicha vista es flotante, es decir, se muestra encima de la anterior Acitividad con un fondo translúcido. Para conseguir este efecto, dentro del *AndroidManifest.xml* que es donde se definen las Activity para que la aplicación las reconozca se le ha aplicado un estilo personalizado para que esta muestre este efecto.

```
<!-- Dentro del AndroidManifest donde se deficen las Activity -->
<activity android:name=".login.FragmentFloatingNewPassword"
    android:theme="@style/FloatingStyleNotTouch"
    android:label="FRAGMENT_NEW_PASSWORD" />
```

Dentro del ficheros de estilos *themes.xml* se ha definido dicho estilo de la siguiente forma.

```
<!-- Extiende de Theme.SafePassword para mantener lo ya definido -->
<style name="FloatingStyleNotTouch" parent="Theme.SafePassword">
    <item name="android:windowContentOverlay">@null</item>
    <item name="android:windowIsTranslucent">true</item>
    <item name="android:windowBackground">@color/translucent</item>
    <item name="android:windowCloseOnTouchOutside">false</item>
    <item name="android:windowTranslucentStatus">true</item>
    <item name="android:windowNoTitle">true</item>
</style>
```

3.4. FragmentFloagingEdit - DeletePassword

Si se desplaza una contraseña hacia la izquierda, comienza el proceso de edición, mientras que si se desplaza a la derecha comienza el proceso de eliminación de la contraseña. Para que ambos casos se lleven a cabo, el usuario, antes de proceder, necesita verificar su identidad. Para este proceso se han creado dos clases, una llamada *PinCode* y *PassCode*. Cuando el usuario realiza una de las acciones anteriores se le preguntará qué método de verificación desea usar. Si elige PIN, se abrirá una ventana-teclado como la siguiente, mientras que si elige CONTRASEÑA, saldrá una caja de texto en la que deberá deintroducirla.

Figura 18: PinCode

Este teclado tiene la característica de mostrar, cada vez que se usa, los números de forma aleatoria. Tanto esta forma de verificación, como la de la contraseña, tiene una forma peculiar de activarse, y es que para llamar a dichas clases (*PINCODE* o *PASSCODE*) se hace uso de los Intent de otra forma. En vez de usar la peculiar manera de llamarlos mediante *StartActivity(...)* esta vez se usará *StartAcitivityForResult(...)*. El porqué de llamar al Intent de esta forma es debido a que al hacerlo así, se le pasa un código a la actividad creada a través del Intent, de forma que en la Actividad nueva, que en este caso será, por ejemplo *PinCode* cuando se haya tecleado el número, esta lo devuelve junto a un código de respuesta (el mismo con el que inició dicha actividad). De esta forma, la que lo llamó puede actuar a razón de lo pedido. En este caso, suponiendo que se llamó para editar la contraseña el proceso que seguiría sería el siguiente.

```
Intent askPin = new Intent(ActivityLogin.this, PinCode.class);
    askPin.putExtra("REQUEST",
    FragmentFloatingPassword.REQUEST_CODE_PIN_EDITPASSWORD);
    startActivityForResult(askPin,
    FragmentFloatingPassword.REQUEST_CODE_PIN_EDITPASSWORD);
```

```
    // Se recoge el codigo por el que dio inicio PinCode
    request = getIntent().getIntExtra("REQUEST", -100);
    ...
    // Tras escribir 4 digitos se convierte a numero y se pasa
    try {
        pin = Integer.parseInt(pin_enter);
        Intent intent = new Intent();
        intent.putExtra("PIN_INTRODUCED", pin);
        setResult(request, intent);
        finish();
    }catch(NumberFormatException nfe) {
        System.err.println("Unable to Parse " + nfe);
    }
```

Se sobreescribe el método *onActivityResult*. Uno de los parámetros que recibe este método es el *requestCode*. Si se construye un *Switch* se puede tratar cada código con una acción pertinente una vez se ha devuelto la contraseña o el pin, según la elección del usuario.

```
case FragmentFloatingPassword.REQUEST_CODE_PIN_EDITPASSWORD:

    code = data.getIntExtra("PIN_INTRODUCED", -100);

    if(code == ActivityLogin.OWNER_PIN) {

        Intent editPass = new Intent(ActivityLogin.this, FragmentFloatingEdit.class);
        // Se le aplica una animacion
        ActivityOptions animacion = ActivityOptions.makeCustomAnimation(
            getApplicationContext(), R.anim.fade_in, R.anim.fade_out);
        //Se inserta un contenido extra al Intent
        editPass.putExtra("PASS_EDIT", passwordSwipe);
        //Se inicializa
        startActivity(editPass, animacion.toBundle());

    }else{
        if(code != -100){
            mensaje.toastAdvice(getString(R.string.pincode_error));
        }
    }

    break;
```

Para finalizar esta sección, al principio de la misma se mencionó que para eliminar o editar una contraseña era necesario desplazar esta hacia izquierda o derecha. Para conseguir esto se ha implementado se ha modificado la acción de los objetos en el *RecyclerView*. Se ha creado un objeto nuevo de tipo *ItemTouchHelper.SimpleCallBack* para poder construir una definición de acciones que permitiesen esto.

```
ItemTouchHelper.SimpleCallback itemTouchHelperCallback = new ItemTouchHelper
    .SimpleCallback(0, ItemTouchHelper.RIGHT | ItemTouchHelper.LEFT) {...}
```

Dentro de este objeto creado, se ha sobreescrito el método *onSwiped(...)* que recibe en uno de sus parámetros un identificador con la dirección del movimiento. Además, con el otro parámetro, es posible saber cual es el elemento que se ha movido.

```
@Override
public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
    // Se conoce el elemento y por tanto se sabe cual es la password
    passwordSwipe = allPassword.get(viewHolder.getAdapterPosition());

    // Se ha desplazado a la derecha
    if(direction == ItemTouchHelper.RIGHT){ // DELETE
        // Se construyen las acciones que se quieran realizar
        // En este caso seria Eliminar una password
        ...

    }else{
        if(direction == ItemTouchHelper.LEFT){
            ...
        }
    }
    rviewListPass.removeViewAt(viewHolder.getAdapterPosition());
    adapter.notifyDataSetChanged();
}
```

3.5. FragmentFloatingPassword

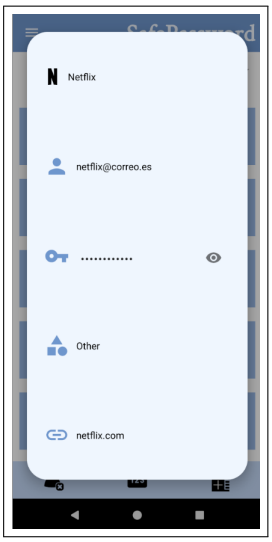

Figura 19: Ver Contraseña

Si en vez de desplazar una contraseña hacia la izquierda o la derecha, se pulsa sobre ella, saldrá una vista como la anterior en la que se podrán visualizar todos los datos de la misma, salvo la propia contraseña. La decisión de esto es para evitar intrusión, y que sea el usuario el único capaz de verla tras una verificación de identidad. Al pulsar sobre el .ºjo.ª la derecha de la contraseña, comenzará el mismo proceso ya explicado en la anterior sección de veficación. Se le preguntará al usuario cómo quiere identificarse, si mediante PIN o CONTRASEÑA. Un detalle no explicado antes, es que si el usuario decide la identificación mediante PIN, pero este no ha sido registrado por primera vez, el proceso que se inicia es el de registrar un nuevo PIN llamando a la misma clase que antes, previa verificación del usuario mediante contraseña. Una vez que la verificación se ha comprobado y todo está correcto, se muestra la contraseña.

Para poner ese elemento con un botón de visualización de la contraseña, en vez de usar un *EditText* se ha usado un *TextInputLayout* al que es posible habilitarle dicho botón. La forma de creación es la siguiente, donde se indica su activación en la línea *app:passwordToggleEnabled* y combinado con *TextInputEditText*.

```
<com.google.android.material.textfield.TextInputLayout
    ...
    app:passwordToggleEnabled="true"

    <com.google.android.material.textfield.TextInputEditText
        ...
        android:focusable="false"
        android:inputType="none|textPassword" />
    </com.google.android.material.textfield.TextInputLayout>
```

3.6. FragmentFloatingBackup

En esta clase se controlan las copias de seguridad que se han generado. Para crear una copia de seguridad se debe hacer desde el botón del menú lateral previamente explicado *Upload Backup*. Al pulsar en este botón, lo primero que ocurrirá es que al Usuario se le preguntarán por los permisos de escritura. En caso de que el usuario acepte se podrá crear un fichero de respaldo en el dispositivo. Al crearse, se pueden acceder a todas las copias de seguridad creadas desde el bóton *Admin Local Backup*. Si al menos hay una, entonces saldrán en modo lista.

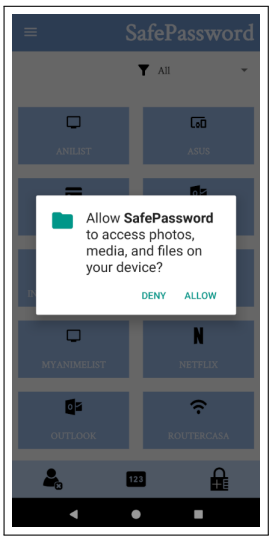

Figura 20: Permisos de Escritura

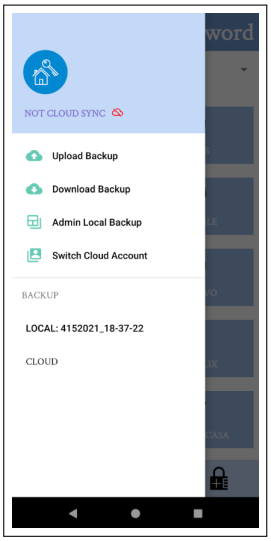

Figura 21: Copia Local

Al pulsar sobre alguna de las contraseñas, se pueden elegir varias opciones. La primera es restaurarla. Al hacer esto, se borrarán las contraseñas actuales almacenadas en el dispositivo y se sustituirán por las contraseñas que estuviesen almacenadas en el momento en el que se hiciese la copia. Otra opción es borrar la copia de seguridad, pero una vez que se haga, no es posible recuperarla. Por último, si el usuario pulsa el botón de borrar de la parte inferior de la pantalla borrará todas las copias de seguridad que haya en el teléfono, sin embargo, el usuario antes deberá identificarse única y exclusivamente mediante la contraseña.

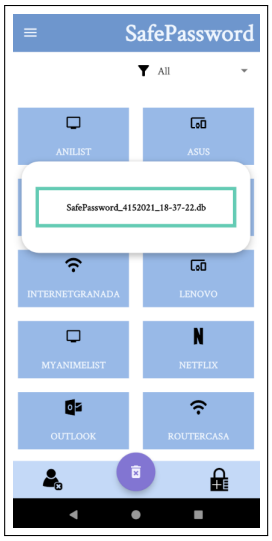

Figura 22: Pantalla de Copias de Sguridad

Figura 23: Opciones Copia de Seguridad

La gestión de la contraseña se hace desde la clase *LocalBackup*. Cuando se pulsa sobre crear copia de seguridad se hace una llamada a esta clase previamente inicializada. Se recoge la localización de la carpeta donde se guardan las copias de seguridad de la aplicación (o de no existir, se crea). Después se crea un fichero con la fecha de creación como nombre. Por último se pasan los bytes del fichero de la Base de Datos actual y se pasan al anterior creados. Si todo este proceso es correcto, entonces se registra en la base de datos, en la tabla de copias de seguridad la fecha de registro de la copia, para que el usuario pueda acceder a ella. A continuación se muestra el código.

```
// SE CREA UNA COPIA LOCAL DE SEGURIDAD EN EL DISPOSITIVO
public boolean doBackup(final DatabaseSafePassword db){
    Context c = activity.getApplicationContext();
    // Se comprueba si el directorio existe
    if (checkDirExist(c)) {
        String f = c.getString(R.string.file_local_backup);
        String dateBackup = mensaje.getDate();
        f = f.concat(dateBackup+".db");
        // Se crea el fichero
        File copyDB = new File(c.getDataDir().toString() + File.separator
            + "databases" + File.separator +
            c.getString(R.string.dir_local_backup), f);
        // Se llama a la base de datos para crear la copia e insertarla
        return db.ExportBackup(c, copyDB, dateBackup, f);

    }else{
        return false;
    }
}
```

```
// Metodo que crea una copia fisica en el telefono de la base de datos actual
public boolean ExportBackup(Context c, File outBackup, String date, String name){

    // Ruta de la base de datos
    final String pathFileName = c.getDatabasePath(DATABASE).toString();

    try {

        // Se recoge el fichero database existente que creo la aplicacion
        File dbFile = new File(pathFileName);
        FileInputStream fis = new FileInputStream(dbFile);

        // Se abre el flujo de salida para poder escribir
            //la base de datos actual al nuevo fichero
        OutputStream output = new FileOutputStream(outBackup);

        // Copia los bytes desde el fichero abierto al fichero de salida
        byte[] buffer = new byte[1024];
        int length;
        while ((length = fis.read(buffer)) > 0) {
            output.write(buffer, 0, length);
        }

        // Se cierran los flujos
        output.flush();
        output.close();
        fis.close();

        // Si llega aqui es que ha ido bien y se puede recoger en la TABLA
        return InsertDataBackup(date, name);

    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

Si por el contrario, el usuario elige restaurar la copia de seguridad, el proceso será el inverso. Se obtiene el nombre del fichero de la base de datos que se creó en su momento. Se eliminan las contraseñas de la base de datos actual y a continuación se abre la base de datos recogida del fichero físico en el dispositivo. Por cada contraseña encontrada en este fichero se inserta en la base de datos actual (pues aunque se restaure, siempre se trabajará con la base de datos creada por la aplicación, siempre se modificará esta). Para no saturar los identificadores, cuando se restaura la copia de seguridad se resetea el número de identificador de las contraseñas.

```
private boolean restoreLocalDatabase(SQLiteDatabase copy){
    if(DeleteAllPasswordDatabase()) {
        // Aunque con openDatabase en modo lectura se supone que ya no es null copy
        if (copy != null) {
            // Se abre una instancia de la Base de Datos actual
            objectDB = getWritableDatabase();

            if(objectDB != null){

                query_text = "SELECT * FROM " + TABLE_PASSWORD
                    + " ORDER BY " + KEY_PASSWORD_NAME + " ASC;";

                Cursor pswCopy = copy.rawQuery(query_text, null);
                if (pswCopy.moveToFirst()) { // Puede ser que tenga 0
                    // Si hay algo se procesa, pero puede haber mas de una
                    do {
                        //Posicion 0 = id
                        //Posicion 1 = name
                        //Posicion 2 = user
                        //Posicion 3 = pass

                        // Cada iteracion del cursor es un elemento Password
                        // Solo falta hacer los INSERT correspondientes

                        INSERT_PASS = "INSERT INTO "+ TABLE_PASSWORD
                        +" ("+KEY_PASSWORD_ID+","+KEY_PASSWORD_NAME
                        +","+KEY_PASSWORD_USER+","+KEY_PASSWORD_PASS+") "
                        +"VALUES (null, '"+pswCopy.getString(1)+"', '"
                        +pswCopy.getString(2)+"', '"+pswCopy.getString(3)+"');";

                        objectDB.execSQL(INSERT_PASS);

                    } while (pswCopy.moveToNext());

                }

                pswCopy.close(); // Se cierra el Iterator
                objectDB.close(); // Se cierra la conexion a la base de datos

            }

            return true;
        }

    }

    return false;
    }
```

```
//Resetea el identificador y empieza en 0 de nuevo
UPDATE_SEQUENCE_PASSWORD = "UPDATE SQLITE_SEQUENCE SET
SEQ=0 WHERE NAME='"+TABLE_PASSWORD+"';";
```

4. Bibliografia

Referencias

[1] Android Developers
https://developer.android.com/

[2] Huella Biométrica
https://developer.android.com/training/sign-in/biometric-auth?authuser=4java/

[3] Criptografia
https://developer.android.com/guide/topics/security/cryptography/

[4] Criptografia
https://arctouch.com/blog/cryptographic-keys-fingerprint-authentication-android/

[5] Google Sign-In
https://developers.google.com/identity/sign-in/android/start-integrating?authuser=1/

[6] Google Drive API V3
https://developers.google.com/drive/api/v3/about-sdk/

[7] Android Developers
https://developer.android.com/

[8] Color Schema Paletton
https://paletton.com/uid=53E0u0k79XU00++23+IbWS7gkIC/

[9] Iconify Design
https://iconify.design/icon-sets/

[10] Example PinView
https://github.com/chinloong/Android-PinView/tree/master/res/