

UNIVERSIDAD DE GRANADA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA Y TELECOMUNICACIONES



Programación de Dispositivos Móviles

MEMORIA TÉCNICA PARA LA APLICACIÓN
SIGN YOUR ID

Jose Manuel Osuna Luque

Índice

1. Introducción	2
2. Desarrollo	3
2.1. Planteamiento	4
2.2. Navegación de la Aplicación	5
2.3. Estructura	6
3. Implementación	7
3.1. ActivityLogin	7
3.2. FragmentActivityWorker	8
3.3. FragmentFirestore	13
4. Bibliografía	17

1. Introducción

En este proyecto se ha construido una aplicación Android enfocada y dirigida a los trabajadores de una empresa. Esta aplicación tiene como funcionalidad y objetivo registrar los turnos y las horas que un trabajador cumple en una empresa, de forma que queden registradas en una Base de Datos. Además de que este pueda registrar un turno, también podrá ver en el calendario los días que ha trabajado, y si pulsa sobre una fecha en concreto aparecerá el turno que hizo con una determinada información. Aun así, todo esto será explicado más adelante.

La aplicación inicia con una interfaz típica, que cuenta con un inicio de sesión donde el trabajador (a partir de ahora se le denominará usuario) deberá introducir su DNI y una contraseña asociada a este para poder iniciar sesión.

Como dato a tener en cuenta, en esta aplicación no es posible registrar a un usuario, ya que al estar enfocada a una empresa, es esta la que debe registrar a un trabajador en su base de datos y proporcionarle unos credenciales para iniciar sesión y que este pueda acceder a sus datos. No obstante, y para la demostración de todas las funcionalidades de la misma, se ha creado un usuario ficticio con el que poder iniciar sesión.

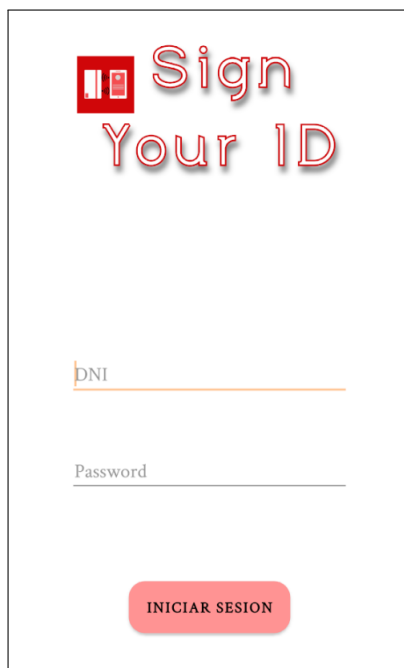
The image shows a login screen for an Android application. At the top, there is a red icon of a document with a checkmark, followed by the text "Sign Your ID" in a red, outlined font. Below this, there are two input fields: the first is labeled "DNI" and the second is labeled "Password". Both labels are in a small, grey font. At the bottom of the screen, there is a red button with the text "INICIAR SESION" in white, uppercase letters.

Figura 1: Login

2. Desarrollo



Figura 2: SignYourID



Figura 3: Icono

SignYourID, tal y como se ha mencionado, es una aplicación enfocada al trabajador de una empresa, de forma que este, al entrar pueda fichar y registrar la hora de comienzo de trabajo, y al finalizar, vuelva a fichar para registrar su hora de salida. Además puede indicar si se trata de un turno único o un turno dividido durante el día. Se debe informar que, esta aplicación es meramente informativa, por la salvedad de que el usuario pueda fichar. El usuario no tendrá la opción de editar sus datos, ni siquiera los turnos en los que haya trabajado, ya que de ese modo, aunque la aplicación esté enfocada a controlar las horas de un trabajador y que este no supere unas horas legales, también debe evitarse que el usuario pueda modificar un registro ya realizado en la base de datos del sistema.

Por ello mismo, se trata de una aplicación sencilla sin muchas más opciones que la de:

- El usuario, podrá iniciar sesión y ver su información detallada
- Podrá registrar una entrada o salida del trabajo.
- Podrá comprobar su historial de trabajo, y al pulsar sobre una fecha donde haya hecho un turno, se le mostrará dicha información

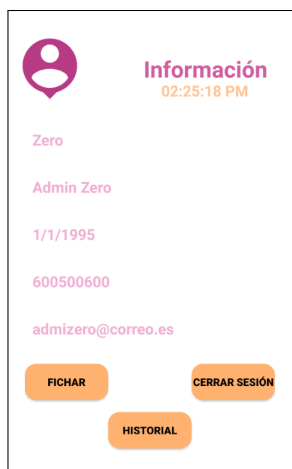


Figura 4: Inicio

Primary color:	#FFB300 #FFEDC3	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300
	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300
	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300	#FFB300 #FFB300

Figura 5: Esquema de Colores

2.1. Planteamiento

Para el desarrollo de la Aplicación se ha elegido el IDE de Google, Android Studio. Además, para la construcción 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 v28. Esto quiere decir que la aplicación desarrollada funcionaría sin problemas en dispositivos de versión 9.0 de Android en adelante.

El por qué de esta decisión se debe a que aproximadamente, alrededor del cincuenta por ciento de los dispositivos android ya usan esta 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, para el almacenamiento de datos, se ha optado por una base de datos en la nube. En concreto, por la Cloud Database de Google, Firebase, más concretamente, el almacenamiento en la nube mediante su servicio Firestore Database.



Figura 6: JAVA



Figura 7: Android Studio



Cloud Firestore

Figura 8: Firebase Firestore

2.2. Navegacion de la Aplicación

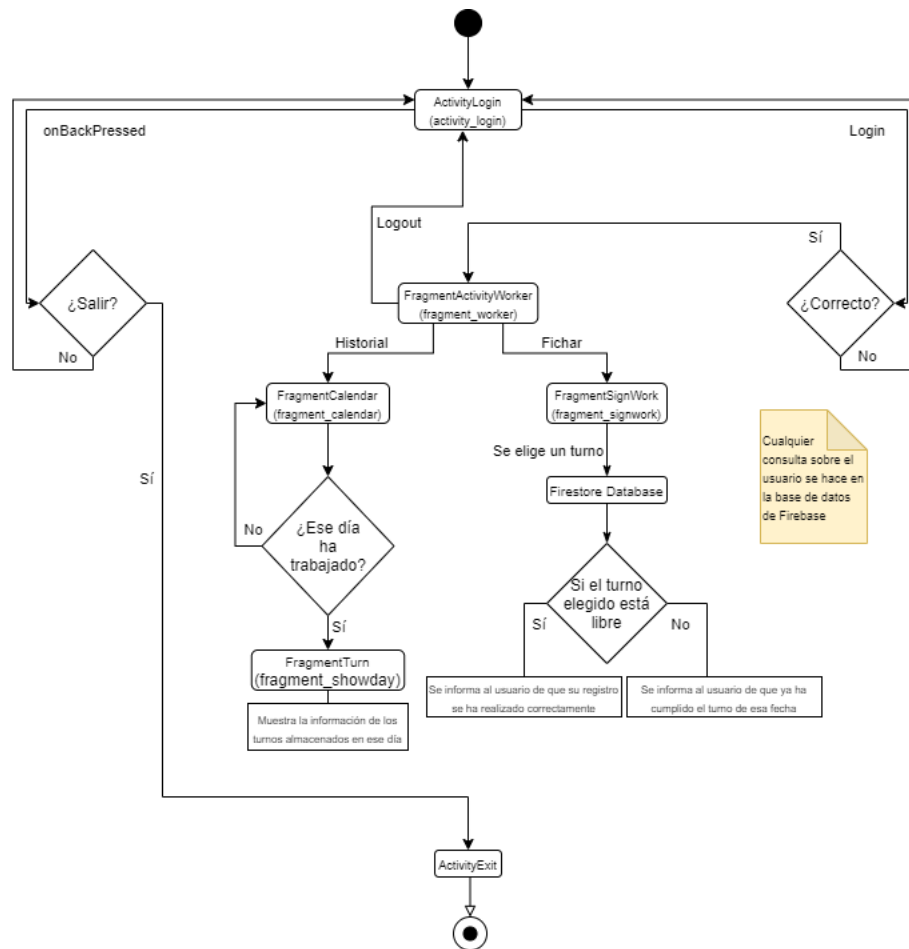


Figura 9: Diagrama de Navegacion

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. Cabe destacar que cada vez que en la aplicación se precisa hacer una consulta sobre los datos del usuario que requieran un acceso a la base de datos, estos accesos se contemplan en las funciones de la clase *Firestore* que será explicada más adelante.

2.3. Estructura

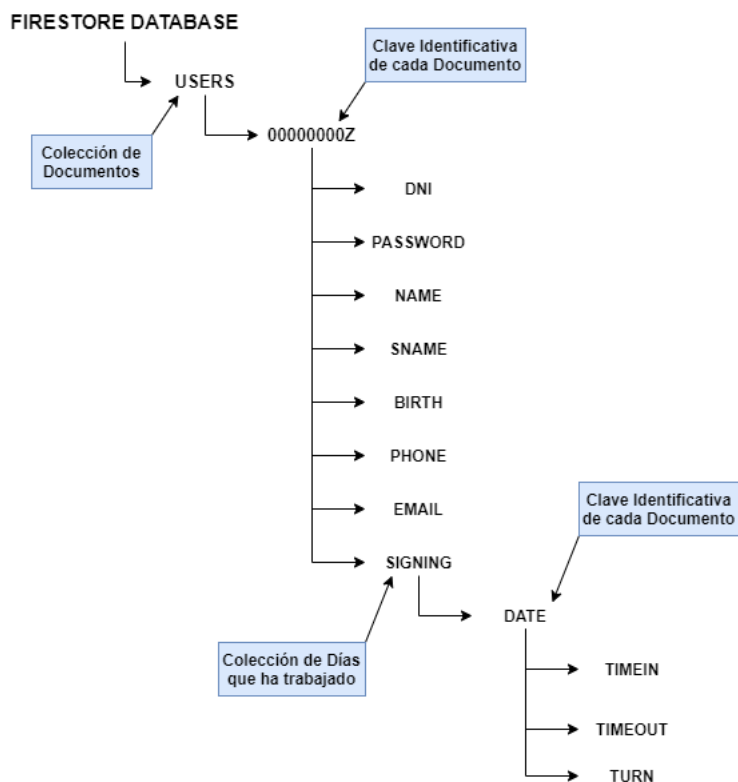


Figura 10: Base de Datos

La aplicación gira en torno a una base de datos en la nube, **Firestore Database**. Esta base de datos tiene una estructura peculiar, y es que se organiza mediante **Colecciones** y **Documentos**, de forma que una **Colección** contendrá **Documentos**, y tanto el uno como el otro están representados por una clave única. Así pues, las **Colecciones** podrían entenderse como **Tablas** en una Base de Datos, y los **Documentos** como el *elemento* almacenado en dicha tabla cuya clave primaria es el **Nombre** de este **Documento**.

La **Colección** **USERS** tendrá un **Documento** por cada usuario registrado; la **Colección** **SIGNING** de cada Usuario contendrá un **Documento** por cada fecha en la que el usuario haya trabajado en la empresa. Los nombres de los **Documentos** debe ser único en una **Colección** pero puede repetirse en **Colecciones** distintas.

3. Implementación

3.1. ActivityLogin

Esta es la primera vista que un usuario verá cuando abra la aplicación, que hará de puente entre las funcionalidades principales de esta misma, de forma que si el usuario no inicia sesión no podrá acceder al contenido de esta.

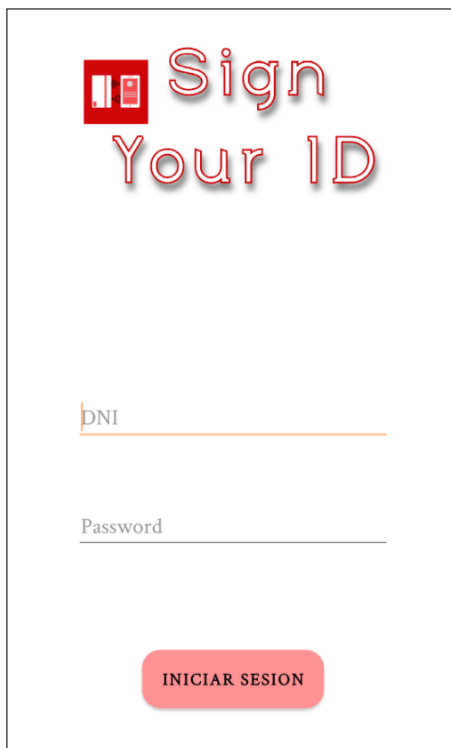
The image shows a mobile application login screen. At the top, there is a red square icon containing a white document with a red checkmark. To the right of this icon, the text "Sign Your ID" is displayed in a large, red, outlined font. Below this header, there are two input fields. The first field is labeled "DNI" in a small, grey font, with a blue underline. The second field is labeled "Password" in a small, grey font, also with a blue underline. At the bottom of the screen, there is a red, rounded rectangular button with the text "INICIAR SESION" in white, uppercase letters.

Figura 11: Activity Login

En ella puede apreciarse el logo de la aplicación, dos campos de texto donde el usuario podrá introducir sus datos de inicio de sesión y un único botón para iniciar sesión. Si el usuario introduce mal sus datos, la aplicación informará con un mensaje de error; sin embargo, si el usuario introduce correctamente sus datos, avanzará al esqueleto de la aplicación donde podrá desarrollar las acciones ya mencionadas.

Cuando el usuario pulsa en el botón, se realiza una consulta a la base de datos mediante la clase *Firestore* (que se explicará más adelante), de forma que si devuelve un dato vacío, quiere decir que el usuario no existe en la base de datos, o que los datos introducidos no fueron correctos.

3.2. FragmentActivityWorker

Es el eje principal de la aplicación, la pantalla de bienvenida al usuario que inició correctamente la sesión. En ella se muestran todos los datos del mismo, nombre, apellidos, teléfono... (salvo la información sensible como puede ser la contraseña). Además, se muestra un reloj en la esquina superior derecha el cual muestra la hora en tiempo real, la foto a la izquierda (no implementado), y una serie de botones en la parte inferior de la pantalla que permitirán al usuario realizar una serie de funciones.

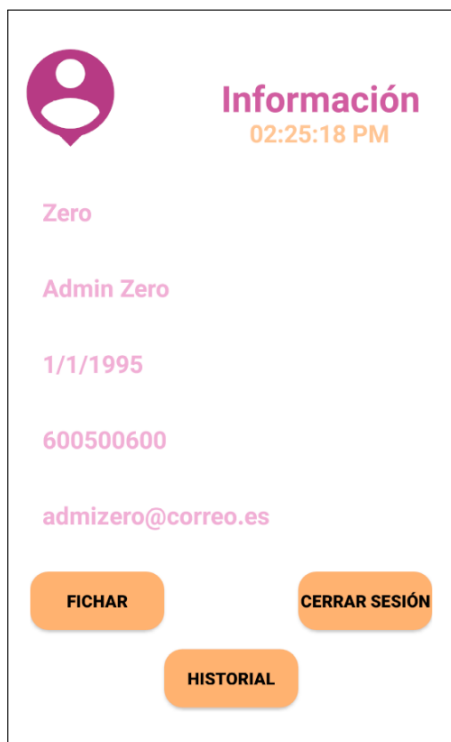


Figura 12: Fragment Activity Worker

Fichar

Es el botón de la izquierda de la parte inferior de la aplicación. Este, quizás, sea la acción con mayor complejidad de la aplicación, ya que para permitir que un usuario pueda fichar en la fecha actual deberán de cumplirse una serie de requisitos para que eso pueda ocurrir. Lo primero que ocurre cuando un usuario pulsa en este botón es que aparecerá un menú desplegable donde el usuario deberá elegir el turno que va a registrar, teniendo que elegir entre tres posibles opciones:

- **TURNO DE LA MAÑANA.** Solo podrá registrarse una hora en dicho turno si el usuario no tiene creada una entrada en la fecha actual, o de tenerla, la hora de salida de dicho turno sigue vacía en la base de datos.
- **TURNO DE LA TARDE.** Solo podrá registrarse una hora en dicho turno si el usuario ya tiene completo el turno de la mañana. Será de entrada al trabajo si ambos campos están vacíos en el turno de la tarde, o de salida en caso de estar completa la hora de entrada.
- **TURNO ÚNICO.** Solo podrá registrarse una hora en dicho turno si el usuario no tiene ninguna otra hora registrada en esta fecha, o, de tenerla, solo puede ser registrada la hora de salida. En caso de que haya un turno único ya registrado en dicha fecha, no podrá añadirse otro.

Turno de Mañana - 1 ▼

REGISTRAR

Figura 13: Registrar

Turno de Mañana - 1 ▼

Turno de Tarde - 2

Turno Único

Figura 14: Turnos Disponibles

Con la explicación anterior, puede hacerse una idea de la estructura de los turnos, de forma que, en una fecha concreta un usuario solo podrá tener almacenados dos posibles opciones: O bien una lista con dos turnos (*Turno de Mañana* o *Turno de Tarde*, o bien un único elemento (*Turno Único*). Una vez se selecciona un turno en el menú, y se pulsa el botón de registro se llamará a la clase que gestiona la base de datos para realizar la acción.

Historial

Tal y como su nombre indica, al pulsar sobre este botón, se abrirá al usuario un calendario donde aparecerá los días que ha trabajado. Esta Activity llamará a una activity auxiliar llamada *FragmentCalendar* que será la encargada de mostrar el calendario posicionado en la fecha actual y donde se recoge la funcionalidad de pulsar sobre un día concreto para hacer una acción.

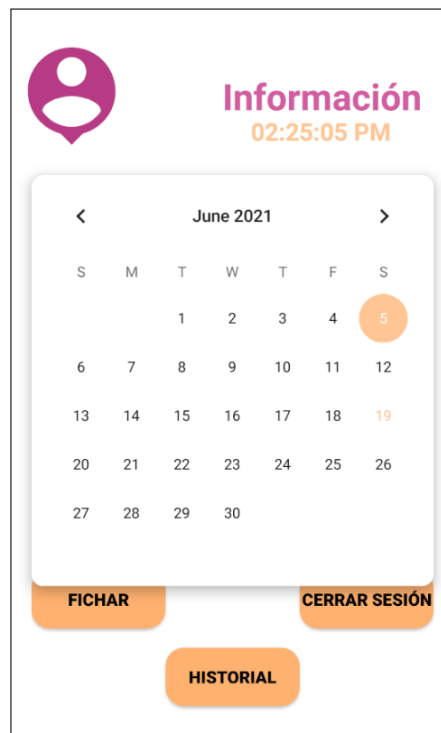


Figura 15: Calendario

Al pulsar sobre un día en concreto, la Activity auxiliar *FragmentCalendar* llamará a la clase que maneja cualquier acción sobre la base de datos para recoger los turnos que un usuario haya realizado el día pulsado. Si encuentra datos se mostrarán en una pestaña.

```
calendarView.setOnDateChangeListener(new CalendarView.OnDateChangeListener() {
    @Override
    public void onSelectedDayChange(@NonNull CalendarView view,
                                    int year, int month, int dayOfMonth) {
        String key = (month+1) + "-" + dayOfMonth + "-" + year;

        ArrayList<WorkDatabase> list = searchDate(key);

        if(list.size() > 0){
            //Llamada a la clase que gestiona la base de datos
            Intent showDay = new Intent(FragmentCalendar.this,
                                       FragmentTurn.class);
            showDay.putExtra("LST-WK", list);
            ActivityOptions animacion =
                ActivityOptions.makeCustomAnimation(getApplicationContext(),
                                                    R.anim.fade_in, R.anim.fade_out);
            startActivity(showDay, animacion.toBundle());
        } else {
            System.out.println("El día " + key + " no hiciste nada, ¿perro?");
        }
    }
});
```



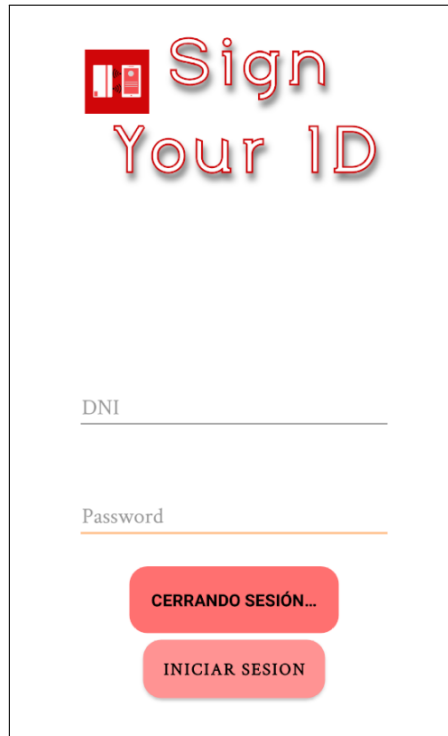
Figura 16: Turno Dividido



Figura 17: Turno Único

Cerrar Sesión

Es el botón más simple. Su única función es enviar al usuario a la pantalla de inicio tras cerrar su sesión y eliminar cualquier rastro de sesión, de forma que si el usuario quiere volver a ver alguna información relacionada con él, deberá volver a introducir sus credenciales.



The image shows a login screen titled "Sign Your ID" in a stylized red font. Below the title, there are two input fields: "DNI" and "Password". The "DNI" field has a red underline, and the "Password" field has an orange underline. Below the input fields, there are two red buttons. The top button is labeled "CERRANDO SESIÓN..." and the bottom button is labeled "INICIAR SESION".

Figura 18: Cierre de Sesión

3.3. FragmentFirestore

Hasta ahora se ha ido indicando qué se podía hacer en cada vista de la aplicación, ya fuese fichar un turno, buscar un día en la base de datos y comprobar si ese día se trabajó para mostrar por pantalla los datos, incluso el inicio de sesión, todos ellos con una serie de requisitos para el avance entre interfaces y muestra de información, sin embargo, ninguna de estas *Activity* mencionadas tienen asociadas dicha funcionalidad, si no que sirven como 'caparazón' a la verdadera funcionalidad interna.

Cada vez que un usuario pulsa un botón o realiza una acción lo que en realidad ocurre es que la aplicación se mueve a otra *Activity* donde se realizan las funciones de lectura y escritura en la base de datos. Esto se consigue mediante los Intent, pero trabajando con ellos de una forma especial, y es activando cada *Activity* con un código asociado, de forma que cuando la *Activity* de **Firestore** finalice las acciones sobre la base de datos sepa a quien debe responder.

```
//Se fija la actividad a la que se navegar
Intent search = new Intent(ActivityLogin.this, FragmentFirestore.class);
// Se añade un código que deber recoger para saber quien lo llama
search.putExtra("REQUEST", REQUEST_FIRESTORE.SEARCHUSER);
// Se añaden los datos que necesitar la clase Firestore
search.putExtra("USRDNL", dni_);
search.putExtra("USRPASS_", pass_);
// Una simple animación
ActivityOptions animacion =
    ActivityOptions.makeCustomAnimation(getApplicationContext(),
        R.anim.fade_in, R.anim.fade_out);
// Forma de llamar a otra actividad con un código identificativo
startActivityForResult(search, REQUEST_FIRESTORE.SEARCHUSER, animacion.toBundle());
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // Si el código coincide con el que envi esta actividad
    if (requestCode == REQUEST_FIRESTORE.SEARCHUSER) {
        // Se recoge el usuario buscado en la base de datos
        UserDatabase us = (UserDatabase) data.getSerializableExtra("USR.RETURN");

        if (!us.getUserName().equals("-n")) {
            // Si el usuario existe y la clave ha coincidido, se inicia sesión
            Intent log = new Intent(ActivityLogin.this, FragmentActivityWorker.class);
            log.putExtra("USR.LOG", us);
            ActivityOptions animacion =
                ActivityOptions.makeCustomAnimation(getApplicationContext(),
                    R.anim.fade_in, R.anim.fade_out);
            startActivity(log, animacion.toBundle());
        } else {
            info.toastAdvice(getString(R.string.not_user_found, dni_));
        }
    }
}
```

El método **startActivityResult()** es la pieza fundamental, ya que esta funcionalidad permite que cuando la actividad a la que se llama finalice, pueda enviar un resultado y la actividad llamadora pueda procesarlo tras sobrescribir el método **onActivityResult**. Cada *Activity* que requiera un acceso a la base de datos deberá construir un *Intent* para hacer una llamada a dicho *FragmentManager* y que este, según el código con el que ha sido llamado, realiza una acción u otra.

No obstante, esto supondría abandonar el hilo de la actividad donde el usuario estaría o incluso cambiarle pantalla. Para evitar esto, se ha hecho uso de la posibilidad de crear *layout* con fondo transparente para una *Activity* simulando una pantalla de carga mientras el *FragmentManager* de **Firestore** está activo realizando las consultas pertinentes. Para generar más aún un efecto de 'carga', el único elemento que contiene la vista de esta *Activity* es una imagen que da vueltas constantemente.



Figura 19: Cargando...

```
loadFlush = findViewById(R.id.loadFlush);
Animation rotate = AnimationUtils.loadAnimation(this, R.anim.rotate);
loadFlush.startAnimation(rotate);
```

```
// Depende del código que reciba esta aplicación realizara una función u otra
switch (request){

    case ActivityLogin.REQUEST_FIRESTORE.SEARCHUSER:

        pass_ask = getIntent().getStringExtra("USRPASS.");
        getUser(dni_ask, pass_ask);

        break;

    case ActivityLogin.REQUEST_FIRESTORE.HISTORIAL:

        getDay(dni_ask);

        break;

    case ActivityLogin.REQUEST_FIRESTORE.SIGN:

        turn_ask = getIntent().getStringExtra("USRTURN.");
        searchDay(dni_ask, turn_ask);

        break;

}
```

Como ya se puede observar en el trozo de código, la *Activity* **Firestore** (*Firestore* asecas, a partir de ahora) podrá recibir tres códigos distintos para los que tendrá que responder correctamente devolviendo dicho código. Es por ello, que lo primero que hace **Firestore** en primer lugar es tomar dicho código y acto seguido generar una instancia, es decir, inicializar una conexión al *Servicio de Google*.

```
//Se coge el codigo
request = getIntent().getIntExtra("REQUEST", -100);

// Siempre se envia el DNI para saber sobre que usuario actuar
dni_ask = getIntent().getStringExtra("USRDNl");

// Se realiza una conexion al Servicio de Google
fire = FirebaseFirestore.getInstance();
```

A partir de este momento, y antes de entrar en detalle en la explicación de los métodos usados en **Firestore** para realizar consultas, debe mencionarse que este servicio de Google trabaja sus peticiones a la Base de Datos de forma **Asíncrona**. ¿Qué quiere decir esto? Que debe controlarse que la *Activity* de *Firestore* no envíe a la actividad que la llamó datos antes de haberse cargado o haberse gestionado correctamente, pues siempre le devolverá un error o un objeto vacío.

Hay dos formas de trabajar con ello. La primera es que al crear una Tarea Asíncrona, tras ejecutarla, se implemente un *Listener* de forma que el método llamado en **Firestore** solo devuelva datos cuando esta consulta se haya realizado correctamente.

```
// Se selecciona el documento de Firestore que se quiere buscar
DocumentReference docRef = fire.collection(tUsers).document(d);

// Se crea un Listener para realizar una accion en caso de que encuentre el objeto
docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
    @Override
    public void onComplete(@NonNull Task<DocumentSnapshot> task) {
        if (task.isSuccessful()) {
            // Entonces se crea el objeto y se devuelve
            returnUser.putExtra("USR_RETURN", userFound);
            //IMPORTANTE DEVOLVER EL MISMO CODIGO CON EL QUE FUE LLAMADA
            setResult(request, returnUser);
            finish();
        }
    }
})
```


La forma anteriormente explicada sirve perfectamente cuando la consulta a la base de datos solo requiere acceder a un único campo, es decir, cuando no hay que acceder al elemento concreto de una colección en el que se haya que comprobar su existencia tras devolver primero una serie de datos y luego recorrerlos. Dicho de otra forma, en el momento que sea necesario tener que realizar varias consultas para comprobar o devolver datos, no bastará con crear un *Listener* ya que estas esperas no son auto-inclusivas, es decir, un *Listener* solo 'esperará' a su tarea, pero si dentro realizas otra tarea que es una consulta a la base de datos, a esta no esperará, y la función seguirá su flujo de vida. Además, si la segunda tarea necesitase de datos de la primera, no es posible asegurar que esta haya finalizado su proceso en segundo plano de forma asíncrona como para que esta los use.

Para este tipo de tareas más complejas se usará una funcionalidad de la clase *Tasks: whenAllComplete()*. De esta forma, se podrá crear una tarea que acceda a la base de datos para obtener unos datos, y el programa sólo avanzará si la tarea indicada ha finalizado. Dentro del *Listener* de *whenAllComplete(TAREA)* se podrá crear otra tarea que use los datos de la anterior y realizar el mismo proceso con *whenAllComplete(TAREA2)* pero sabiendo que llegados a este punto ya tenemos los datos de la anterior consulta. Teniendo en mente esta estructura, se pueden crear consultas complejas.

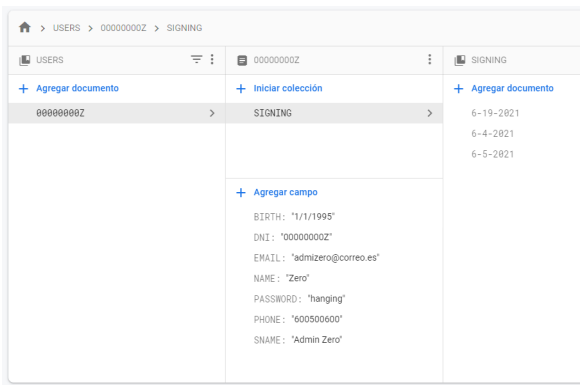


Figura 20: Estructura Firestore

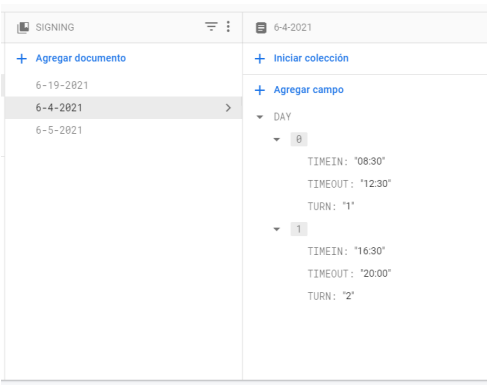


Figura 21: Estructura Firestore

4. Bibliografia

Referencias

- [1] Android Developers
<https://developer.android.com/>
- [2] Firebase
<https://firebase.google.com/>
- [3] Stackoverflow
<https://stackoverflow.com/>
- [4] Color Schema Paletton
<https://paletton.com/>