DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES

Grado en Ingeniería Informática en Tecnologías de la Información – 2015-16

Práctica 2

Gestos y almacenamiento

1. Objetivos fundamentales:

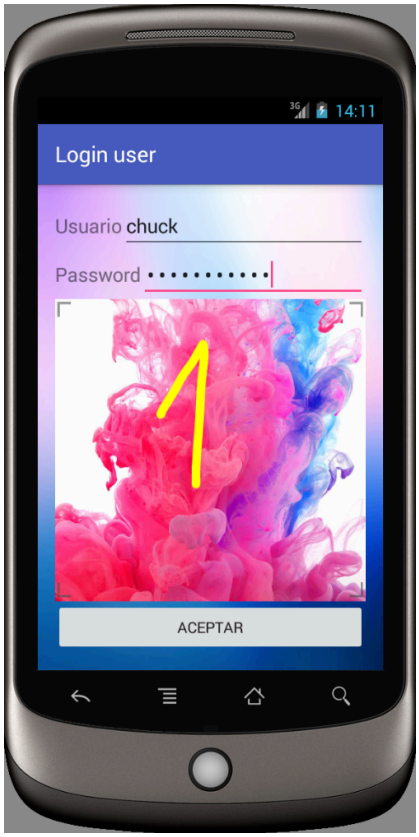
- Estudio del manejo de elementos gestuales.
- Aprender los conceptos básicos del almacenamiento persistente de ficheros (interno / externo) en la plataforma Android.
- Manejar el uso de SQLite como gestor de base de datos relacional en las plataformas.
- La práctica se entregará a través de la web de la asignatura en su correspondiente apartado de tareas. Se deberá entregar en un archivo comprimido. El proyecto no podrá exceder de 20 megas. Todos aquellos archivos que sean detectados como virus no serán corregidos. La fecha límite de entrega será el **17 de Mayo del 2016 a las 23:55h**.

2. Descripción de la práctica.

Se trata de realizar un aplicativo para el almacenamiento de logins y passwords de diferentes aplicaciones, tanto móviles como web. Es decir, mediante un usuario, un password y un gesto accederemos a nuestro almacén de logins (usuario, password) y otros datos de todas nuestras aplicaciones.

Lo primero que se visualizará será una pantalla de login, donde se solicitará usuario, password y gesto.

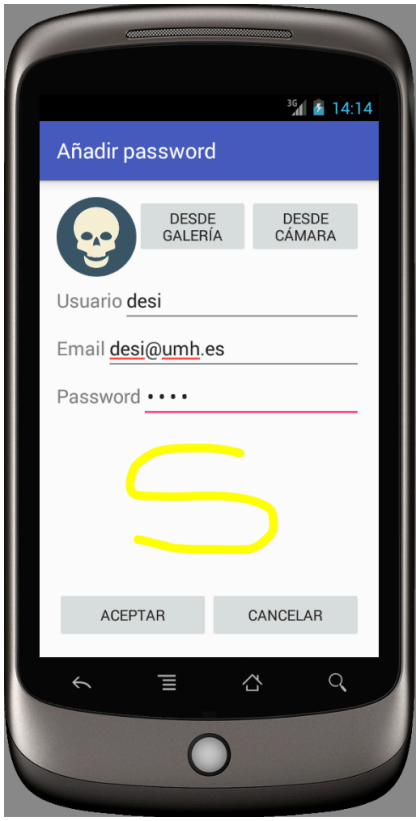
Pantalla de ejemplo:



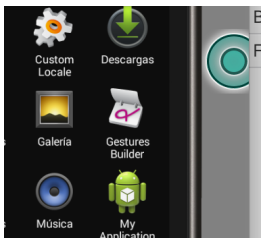
Evidentemente la primera vez que arranquemos la aplicación no existirá ningún usuario dado de alta, por lo tanto, se creará un menú con la opción **Añadir usuario** que mostrará una pantalla para la introducción de los datos básicos del usuario:

- Imagen (Se podrá obtener desde la galería de imágenes o mediante la realización de una foto, para ello se incorporarán 2 botones distintos)
- Usuario (alias)
- Correo
- Password
- Gesto que posteriormente tendremos que introducir.

Pantalla de ejemplo:



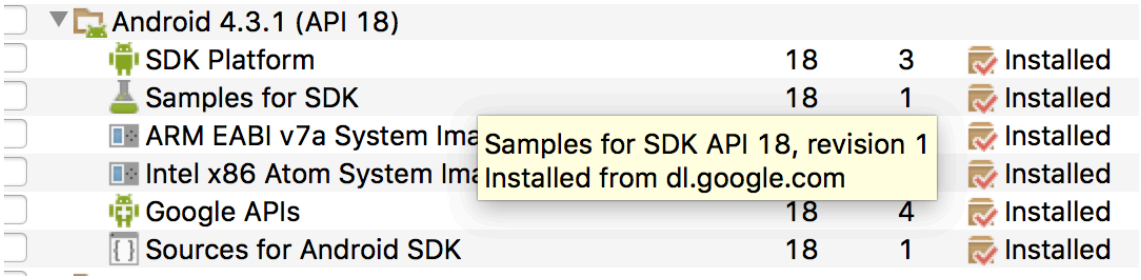
Para implementar la parte de los gestos nos basaremos en el código generado por una aplicación existente en el emulador. La aplicación Gestures Builder permite almacenar gestos con un nombre en un almacén de gestos.



Podemos encontrar su código fuente en la siguiente ruta:

ruta de SDK/samples/android-18/legacy/GestureBuilder

Ojo que para tener el código fuente, antes debemos tener instalado el paquete **Samples for SDK**



Basándonos en ese código, y en el que se proporcionará en ejemplos de clase de prácticas, se realizará el almacenamiento y la validación de los gestos.

Tras introducir los datos correctos, se deberá almacenar el usuario y futuros usuarios mediante algún método de persistencia de datos. En este caso utilizaremos la memoria interna del dispositivo (ya que es privada de la aplicación). Se creará un ArrayList de objetos de la clase usuario y este se persistirá en un fichero de la memoria interna. La serialización del ArrayList de usuarios se deja a libre elección, siendo más que recomendable la utilización de una librería externa, como por ejemplo GSON de Google que persiste objetos de Java a formato JSON de forma transparente para el usuario.

```
// Serialization
Gson gson = new Gson();
String objJSONSerialized = gson.toJson(arrayUsers);

// Deserialization
Gson gson = new Gson();
return gson.fromJson(cadena.split("\\n")[0], UserPersist.class);
```

Desde la pantalla inicial, en el caso de introducir un usuario, password y gesto correcto se mostrará otra pantalla con los logins dados de alta para ese usuario.

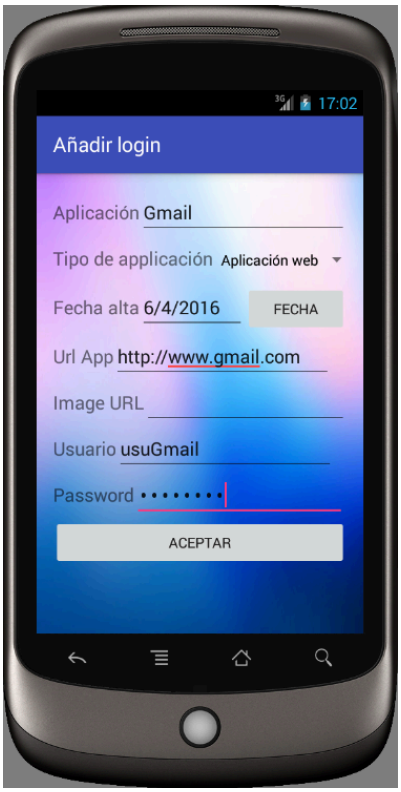
Pantalla de ejemplo de listado de logins vacía:

Para almacenar los logins utilizaremos una bd con el motor de bd SQLite. Para ello, haréis uso de la creación de una base de datos de tipo SQLite con cualquiera de las opciones de software externo que se ha visto en clase, durante el curso, o creando las tablas desde código (mi recomendación, es que utilicéis este último modo), que contendrá una tabla con los siguientes campos:

- Tabla llamada **Logins**, y 9 campos **_id** (será INT, PRIMARY y AUTO INCREMENT), **alias** (text), **loginType** (text), **nameOfApp** (text) , **urlApp** (text), **imageURL** (text), **user** (text), **password** (text) y **fecha** (long)
- En el caso de la imagen, se mostrará una almacenada por defecto en la aplicación, salvo que se opte por la ampliación que existe en la parte opcional.
- La pantalla con el listado de los logins cargados de la aplicación tendrá un aspecto similar a este:

- Para introducir películas, usaremos el menú que surge cuando se pulse en el botón Menú de los smartphones Android, y contendrá un submenú con las siguiente opción y aspecto:
  - Añadir login.

- Cuando se pulse en **Añadir login**, nos llevará a la actividad que nos servirá para introducir los valores del login a guardar. Tendrá el siguiente aspecto:



- Los control Spinner tipoAplicación tendrá los siguientes valores:
  - Aplicación web y Aplicación móvil
- Implementar la posibilidad que cuando el usuario realice una pulsación corta sobre un ítem de la lista, se mostrará de nuevo el formulario de inserción, con los datos del registro correspondiente para poder realizar modificación de datos.
- Si el usuario realiza una pulsación prolongada sobre el ítem de la lista, aparecerá la opción de eliminar dicho registro mediante un menú contextual al elemento seleccionado. Habrá que seguir los preceptos de usabilidad y ante una opción de borrado, preguntar al usuario si desea realmente realizar dicha operación.
- Como hemos comentado, para salvaguardar los datos de los logins, deberemos utilizar una base de datos SQLite llamada BDLogins.db y acceder a su información usando las funciones vistas en clase durante el

curso. La creación de la base de datos en código la podremos realizar con la siguiente sentencia:

```
"CREATE TABLE Logins (_id integer primary key autoincrement,
alias text not null, loginType integer not null, nameOfApp text
not null, urlApp text, imageURL text, user text not null,
password text not null, firstdate long)"
```

A tener en cuenta

La visualización de las pantallas es meramente indicativa, la apariencia se puede cambiar. Se valorará positivamente el diseño alternativo de las pantallas.

Es obligatorio utilizar los ficheros de recursos para layouts, cadenas, imágenes, etc…

PARTE OPCIONAL

- Personalizaciones de la práctica como por ejemplo:
  - Agregar nuevos campos.
  - Añadir nuevas pantallas.
  - A partir de la URL de la aplicación y si esta es de tipo web, permitir añadiendo una nueva opción al menú contextual del elemento (junto al borrar elemento) para poder abrir el navegador y que abra directamente la aplicación para poder hacer login.
- Se valorará la utilización de librerías de terceros para:
  - Obtención de las imágenes de los logins a partir de la imageURL: Se descargará la imagen a partir de esa URL y se visualizará en el listado. Se podrá utilizar librerías de terceros como **Picasso**.
  - Facilitar el desarrollo de aplicaciones Android como por ejemplo:
    - ButterKnife para establecer el mapeo del layout a las clases de Java mediante anotaciones. En este enlace (http://blog.teamtreehouse.com/android-libraries-use-every-project) podéis encontrar algunas librerías y ejemplos de las mismas.

3. Evaluación

Almacenamiento de Usuarios en memoria interna y pantallas de login y alta de usuario → 3 Puntos

Almacenamiento de Logins en SQLite y pantallas de listado de logins de un usuario → 5 Puntos

Opcional → 2 Puntos

AVISO: En el caso de que la asistencia del alumno no sea de al menos el 75% de las clases de prácticas, será obligatoria la entrega de una memoria (2 o 3 páginas) justificando las tareas clave realizadas de la misma, y en su caso, se deberá defender en una tutoría.

ANEXO I . ALMACENAMIENTO FICHEROS INTERNO y SDCARD

Android nos provee con diversos métodos de almacenamiento persistente de los datos manejados en las aplicaciones. Entre estos métodos destacan los siguientes:

- Manejo de ficheros en la **memoria interna** del dispositivo. Por defecto, estos ficheros son privados a la aplicación que los crea, ninguna otra aplicación tiene acceso a ellos. La desinstalación de la aplicación provoca el borrado de dichos ficheros.
- Manejo de ficheros en la memoria externa del dispositivo. Tales como tarjetas SD, MMC, etc.

Empezaremos practicando sobre el tipo de almacenamiento interno y externo de ficheros.

Poniendo toda nuestra atención en el método de almacenamiento interno, comenzaremos realizando una práctica sencilla en la que comprobaremos dicha funcionalidad.

Partiendo de las dos alternativas que tenemos en almacenamiento interno, la primera con métodos de la librería de I/O de Java ,y la segunda ,con métodos propios de Android, estudiaremos esta última por su facilidad de uso.

Android para facilitar esta tarea nos provee del método openFileOutput(), que recibe como parámetros el nombre del fichero y el modo de acceso con el que queremos abrirlo. Los diferentes modos de acceso que tiene este método implementados son los siguientes:

Modo	Descripción
MODE_APPEND	Si el fichero existe, añadir contenido al final del mismo. Si no existe, se crea
MODE_PRIVATE	Modo por defecto donde el fichero sólo puede ser accedido por la aplicación que lo crea
MODE_WORLD_READABLE	Permite a las demás aplicaciones tener acceso de lectura al fichero creado
MODE_WORLD_WRITEABLE	Permite a las demás aplicaciones tener acceso de escritura al fichero creado

Utilizando el método OutputStreamWriter nos permitirá escribir una cadena de texto al fichero. El siguiente código nos muestra su uso:

```
try
{
    OutputStreamWriter fout=
        new OutputStreamWriter(
            openFileOutput("fichero_interno.txt",
     Context.MODE_PRIVATE));

    fout.write("Introduccion de texto en el fichero.");
    fout.close();
}
catch (Exception ex)
{
    Log.e("Ficheros", "Error al escribir fichero a memoria
interna");
   }
```

Como se indicó en teoría si deseamos poder visualizar la creación de nuestro archivo, una vez ejecutado el código deberemos activar la perspectiva DDMS y buscar dicho fichero en la ruta siguiente:

```
/data/data/paquete_java/files/nombre_fichero
```

Para leer ficheros tan sólo tendremos que indicar al método openFileInput () que lo vamos a abrir para lectura utilizando los método propias de la librería java.io:

```
try
{
    BufferedReader fin =
        new BufferedReader(
            new InputStreamReader(
```

```
                        openFileInput("prueba_int.txt")));

            String texto = fin.readLine();
            fin.close();
        }
        catch (Exception ex)
        {
            Log.e("Ficheros", "Error al leer fichero desde memoria
        interna");
              }
```

Pasemos a ver, a continuación, los métodos de acceso de almacenamiento en memoria externa. Una buena práctica de programación para este uso de almacenamiento, consiste como primer paso ineludible en comprobar la disponibilidad de dicha memoria externa. Con el método **getExternalStorageState()** de la clase Environment tendremos resuelto el problema. Veremos como comprobar dicho estado en el siguiente código:

```
//Creamos dos variables booleanas que nos permitirán registrar los
estados de la memoria
boolean mExternaHabilitada= false;
boolean mExternaEscribible= false;
//Crearemos una string para guardar el estado de dicha memoria
String estadoMemoria = Environment.getExternalStorageState();
//Comprobaremos si podemos leer y escribir en la memoria externa
if(estadoMemoria.equals(Environment.MEDIA_MOUNTED)){
      mExternaHabilitada = true;
      mExternaEscribible = true;
} //Podremos leer pero no escribir
else if (estadoMemoria.equals(Environment.MEDIA_MOUNTED_READ_ONLY)){
            mExternaHabilitada = true;
            mExternaEscribible = false;
else
//No se puede ni leer ni escribir
      {
            mExternaHabilitada = false;
            mExternaEscribible = false;
      }
```

Una vez comprobada la disponibilidad usaremos el método getExternalFilesDir () de la

clase Context para abrir el fichero en cuestión. Si deseamos guardar archivos en la memoria externa que no queremos que sean borrados cuando la aplicación se desinstala, tan sólo deberemos guardar dichos ficheros en los directorios públicos que cuelgan de la memoria externa como pueden ser Music/, Pictures/, Ringtones/, etc.

ANEXO II. SQLite

La base de datos SQLite es un motor de base de datos que cada día tiene mejor aceptación entre la gran comunidad de desarrolladores. Entre sus excelencias, destacan por encima del resto, que no necesita un servidor, su configuración es extremadamente sencilla y simple, y el tamaño es minúsculo. Si a todo eso le añadimos que es código abierto, estamos ante una oferta realmente tentadora a la hora de implementar nuestros desarrollos.

Esta herramienta se basa sobre la clase SQLiteOpenHelper y su función no es otra que la de comportarse como un plantilla sobre la que personalizaremos nuestra base de datos.

Si creamos una clase SQLiteOpenHelper estamos obligados forzosamente a implementar los siguientes métodos:

- onCreate (SQLiteBBDD)
- onUpgrade (SQLiteBBDD, int, int)

onOpen(SQLiteBBDD) está última con carácter opcional.

Deberemos de hacer uso del patrón de diseño visto en clase y crear una clase **Adaptador** en donde se encontrará contenida la clase **SQLiteOpenHelper**. Su función no es otra que la de comportarse como un plantilla sobre la que personalizaremos nuestra base de datos.