POLITÉCNICA

Agenda

ANDRÉS MICÓ MILLÁN

Máster universitario en desarrollo de aplicaciones móviles y servicios para dispositivos móviles

Tabla de contenido

1. Introducción

La aplicación calculadora para Android esta optimizada para dispositivos de 5 o más pulgadas funcionando a partir de la API 19 o 4.4 (KitKat), tanto en ***Portrait***, como en ***LandsCape***. Está compuesta por diversas actividades:

- *MainActivity*, que tiene gran parte de la lógica de las funcionalidades implementadas en la agenda.
- *BDAgenda*, clase que extiende SQLiteOpenHelper proporcionándonos una BD local para almacenar nuestros contactos.
- *Contacto,* clase que representa un contacto, es decir crear una estructura que nos permite gestionar sus atributos,
- *Adaptador,* clase que extiende CursorAdapter permitiendo modificar y personificar el contenido de las filas de un ListView.
- *ModiContacto,* actividad que nos muestra la información de un contacto, permitiéndonos modificar o actualizar su información.
- *VerContacto,* actividad que nos muestra la información sobre un contacto, permitiéndonos llamarlo por teléfono.
- *SplashAct,* actividad de información sobre la app.

La aplicación pretende ser una agenda de contactos, pudiendo añadir, modificar, eliminar contactos, así como, llamarlos por teléfono. Además, debe de implementar varias funcionalidades que dan valor añadido a la app.

2. Funcionalidad

En este apartado comentaremos las funcionalidades más destacadas de la Agenda desarrollada para Android:

- Crear, modificar, eliminar, buscar contactos y llamar a contactos.
- Posibilidad de añadir una imagen para los usuarios.
- Datos de los contactos almacenados de forma local usando SQLite.
- Importar y exportar los contactos a un archivo .json o .txt
- Menú contextual asociado a mantener pulsado un elemento de la lista.
- Uso de la sd y gestión de ficheros.
- Gestión de permisos para dispositivos con API superior a 23.
- Uso de strings y multilenguaje.

En cuanto al diseño la aplicación incorpora elementos de Material Design como:

- Cuenta con un Bottom Sheet que, al pulsar un contacto de la lista, se despliega de la parte inferior un panel deslizante con la información del usuario seleccionado.

- Toolbar personalizada en la parte superior de vista que contiene un EditText para escribir el nombre del contacto que queremos buscar, además de un botón con un menú desplegable con las opciones de Importar y Exportar.
- NavigationView personalizada que se despliega pulsando un botón situado en la toolbar o deslizando el dedo desde la parte izquierda de la pantalla hacia el centro. La NavigationView cuenta con dos opciones para poder ordenar de distinta forma nuestra lista de contactos.
- TextInputLayout, que nos permite incorporar mecanismos de usabildiad a la aplicación, de tal forma que podemos indicar al usuario si está introduciendo unos valores incorrectos para un determinado campo en tiempo real. Además, mejora la UX ya que los EditText tienen, entre otras cosas, animaciones.

Además, en cuanto funcionalidad podemos decir que tiene ***SnackBars*** de información por si el usuario comete algún error o para informar al usuario de las acciones que va realizando. También cuenta con ***Dialogs***, para gestionar puntos críticos como el botón de cerrar la aplicación, o para elegir el formato a la hora de exportar o importar los contactos. Así mismo, cuenta con una interfaz desarrollada para el modo de ***LandsCape***, permitiéndonos poder girar el dispositivo de forma horizontal. Por último, cuenta con una ***SplashScreen,*** y animaciones entre vistas.

3. Métodos

En este apartado se explicará de una forma más detallada el código que se considera más importante, además de tratar algunos métodos que incorporan lógica.

Empezando por la clase BDAgenda, nos encontramos con la Base de Datos que sigue la siguiente forma:

```
"CREATE TABLE contactos " +
        "(_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +
        " nombre VARCHAR(100), direccion VARCHAR(100), movil VARCHAR(100)," +
        " mail VARCHAR(100));";
```

Figura 1: Tabla Contactos SQL

Podemos observar los campos *_id* que contendrá el identificador único del contacto, *nombre, dirección, móvil* y *mail* del contacto, todos ellos considerados como una cadena. Cabe destacar que las imágenes son guardadas en la SD del dispositivo teniendo el siguiente formato *nombre_del_contacto.png*, esta imagen se modifica cada vez que el usuario modifica su nombre o la propia imagen de contacto.

En cuanto funcionalidades destacables nos encontramos con el siguiente método qué dado el nombre de un contacto, devuelve toda la información almacenada en la BD.

```java
// Devuelve datos de un Contacto dado un nombre, los devuelve de forma orednada por nombre
public Cursor busquedaContacto(String nombre) {

    if(nombre.length() > 0){
        SQLiteDatabase db = getReadableDatabase();

        String[] valores_recuperar = {"_id", "nombre", "direccion", "movil", "mail"};

        String[] args = new String[] {nombre};

        Cursor c = db.query("contactos", valores_recuperar, "nombre=?", args, null, null,
                "nombre ASC",null);
        return c;

    }
    else {
        SQLiteDatabase db = getWritableDatabase();

        String[] valores_recuperar = {"_id","movil","nombre"};

        // Ordena al recuperarlos
        Cursor c= db.query("contactos",valores_recuperar,null,null,null,null,"nombre ASC",null);

        return c;
    }
}
```

Figura 2: Búsqueda de la información un contacto

En la siguiente imagen podemos ver el código asociado al método que recoge la imagen después de que el usuario seleccione la imagen desde la galería.

```java
// Recogemos la imagen elegida por el usuario
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data)
{
    // Si ha elegido una imagen
    if (requestCode == PICK_IMAGE) {
        try{
            // Si tenemos datos
            if (data != null){

                // Obtenemos una URI de la ubicación de la imagen
                imageUri = data.getData();

                // Abrimos la URI a través de un InputStram
                final InputStream imageStream = getContentResolver().openInputStream(imageUri);

                // Convertimos ese inputstream a un BitMap
                selectedImage = BitmapFactory.decodeStream(imageStream);

                // La asignamos a la ImagevIEW
                image_foto.setImageBitmap(selectedImage);

            // Si no hay data significa que el usuario no ha elegido ninguna imagen
            }else{
                setSnackBar(mLayout,"No has elegido ninguna imagen");
            }

        } catch (IOException e) { e.printStackTrace(); }
    }
}
```

Figura 3: Recogemos la imagen y la asignamos a la ImageView

Los métodos asociados a importar y exportar contactos nos permiten guardar en la tarjeta SD la información asociada a nuestros contactos, tanto en formato texto, como en formato Json.

El método de exportar los datos a Json tiene la siguiente forma:

```java
// Crea el fichero Contatos.json con los datos obtenidos de la lista
public void crearFileJson(){

    // Establecemos la ruta donde guardar el archivo .JSON
    File ruta = Environment.getExternalStorageDirectory();
    // Creamos el archivo
    File file = new File(ruta.getAbsolutePath(), "Contactos.json");
    // Creamos un JSArray
    JSONArray contactos = new JSONArray();
    // Necesario para acceder a los atributos del contacto
    Contacto contacto;

    try{
        // Para cada registro de la lista, añadimos sus valores en el objeto Json
        // y lo añadimos al array previamente creado.
        for (int i=0; i < getListView().getCount(); i++){
            JSONObject root = new JSONObject();
            contacto= bdAgenda.recuperarContacto(ids[i]);
            root.put("nombre",contacto.getNombre());
            root.put("movil",contacto.getMovil());
            root.put("direccion",contacto.getDireccion());
            root.put("mail",contacto.getMail());
            contactos.put(root);
        }

    }catch (Exception e){}

    JSONObject contacto_final = new JSONObject();
    try{ contacto_final.put("Contactos",contactos); }catch (Exception e){}

    // Abrimos el archvio como y añadimos la cadena en formato JSON
    try {

        OutputStreamWriter f_out = new OutputStreamWriter(new FileOutputStream(file));
        f_out.write(contacto_final.toString());
        f_out.close();
        setSnackBar(mLayout,"Los datos fueron exportados a Contactos.json correctamente");

    } catch (Exception e) {setSnackBar(mLayout,"No hemos podido exportar el archivo en formato .json");}
}
```

Figura 4: Exportar contactos a un archivo .json

Es por ello que la estructura de Json tiene una estructura como la siguiente, donde cada elemento del array representa un contacto.

```
{"Contactos":
  [
    {
        "nombre":"Andres",
        "telefono":"123456789",
        "xxx":"xx",...
    },
    {   "nombre":"xx",
        "..."
    }
  ]
}
*/
```

Figura 5: Estructura del archivo Contactos.json

Por otro lado, existe el método ***importarFileJson***, que lee los contactos del archivo los interpreta y los muestra en pantalla.

```
// Recupera los contactos del fichero Contaxtos.json
public void importaFileJson(){

    File ruta = Environment.getExternalStorageDirectory();
    File file = new File(ruta.getAbsolutePath(), "Contactos.json");

    // ArrayList para obtener los registros del File y añadirlos a la BD
    ArrayList<String> nombreFromFile =  new ArrayList<>();
    ArrayList<String> movilFromFile =  new ArrayList<>();
    ArrayList<String> direccionFromFile =  new ArrayList<>();
    ArrayList<String> mailFromFile =  new ArrayList<>();

    // Si no encuentra el archivo
    if (!file.exists()) {
        setSnackBar(mLayout, "No hay ningun archivo .json para importar");

    } else {
        try {

            // Borramos los contactos actuales
            for (int i=0; i < getListView().getCount(); i++) {
                bdAgenda.borraContacto(ids[i]);
            }

            // Clase para leer lineas desde un archivo en Java
            BufferedReader f_in = new BufferedReader(new
                    InputStreamReader(new FileInputStream(file)));

            // Pasamos el contenido del archvio a un string
            textoLecturaFin = f_in.readLine();

            // Parseamos el contenido a JSON
            JSONObject root = new JSONObject(textoLecturaFin);
            JSONArray jsonArray = root.getJSONArray("Contactos");

            for(int i=0; i<jsonArray.length(); i++) {

                // Obtenemos el objeto contacto y los diferentes campos
                JSONObject json_data = jsonArray.getJSONObject(i);

                nombreFromFile.add(json_data.getString("nombre"));
                movilFromFile.add(json_data.getString("movil"));
                direccionFromFile.add(json_data.getString("direccion"));
                mailFromFile.add(json_data.getString("mail"));

            }

            // Insertamo los nuevos campos
            for (int j=0; j<nombreFromFile.size() ;j++){
                bdAgenda.insertarContacto(nombreFromFile.get(j),direccionFromFile.get(j),
                        movilFromFile.get(j),mailFromFile.get(j));
            }

            f_in.close(); // Cerramos el archivo
            rellenaLista(); // Cargamos de nuevo la vista con los datos importados
            setSnackBar(mLayout,"Contactos.json importado correctamente");

        } catch (Exception e) {}
    }
}
```

Figura 6: Método que importa los datos de los contactos desde un .json

Por último, veremos el almacenamiento del dispositivo, comprobando que los dos archivos se crean en la SD del dispositivo.

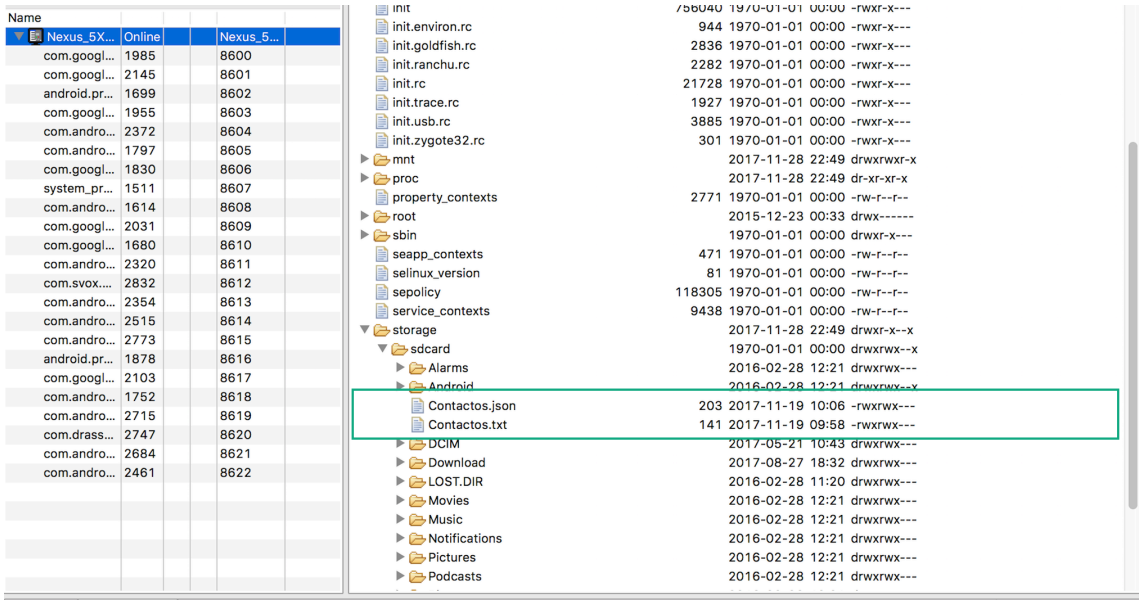

Figura 7: Archivos guardados en la SD.

4. Conclusiones

Gran parte del código está documentado, tantos las funciones como los métodos y las variables que se utilizan. A continuación, hay un enlace a un video que muestra las funcionalidades de la aplicación y como sería el uso por parte de un usuario real.

https://mega.nz/#!lsISkbob!CtaPEpK6LfiuFailfdUJi-RAQTsa9CbMgjGaX1EYDKc