Licenciatura en Sistemas - Taller de Aplicaciones Móviles – 2022

Bibliografía: https://developer.android.com/

Equipo docente: Lic. Romina Mansilla

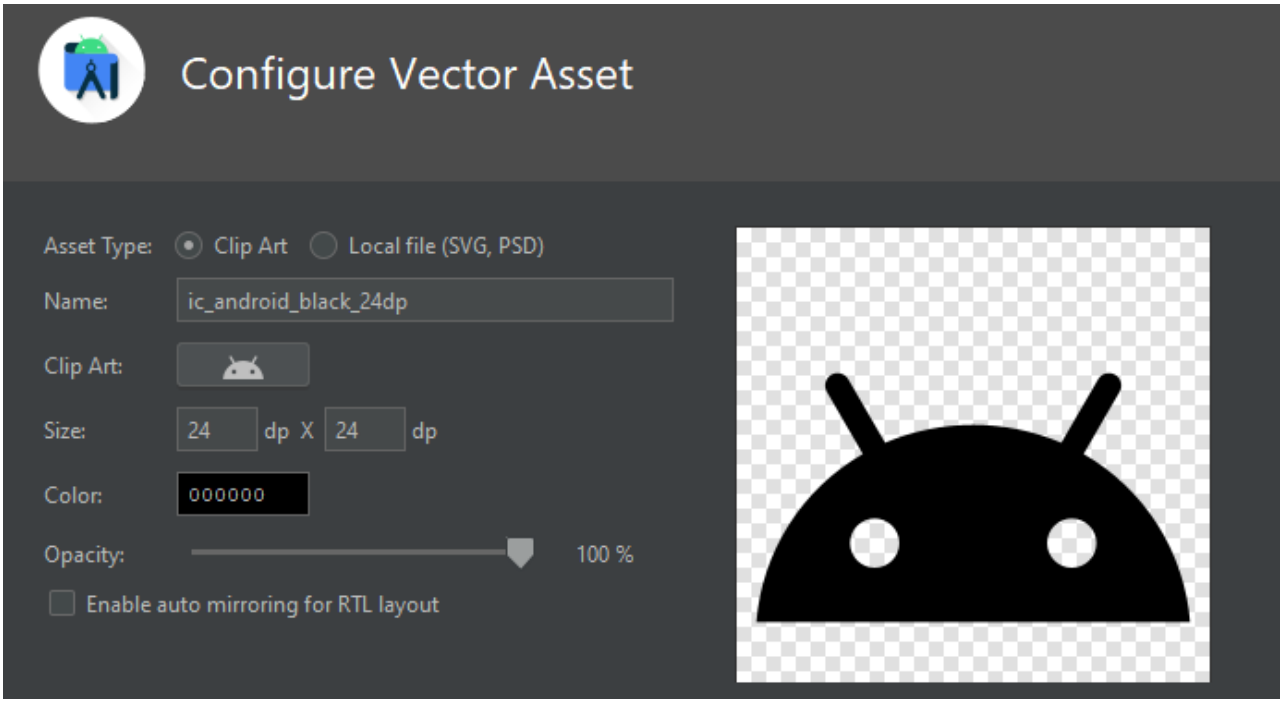
Unidad 3

Imágenes nativas Android Studio.

Iconos

El Android Studio provee una herramienta que permite agregar imágenes precargadas que el IDE contiene. Estas están categorizadas en íconos para la Action Bar, para las notificaciones y para el launcher de la aplicación.

Para acceder a esta funcionalidad, se deberá posicionar sobre la carpeta res -> drawable, click derecho, new -> Vector Asset.



Presionando sobre la imagen del Android del Clip Art traerá una galería de imágenes que se pueden seleccionar. Luego de elegir, se la deberá nombrar y cambiarle el color al que se desee.

Una ventaja de utilizar esta herramienta es que por defecto genera la imagen en formato Vector, por lo que esta se verá correctamente en distintos dispositivos con distintas resoluciones.

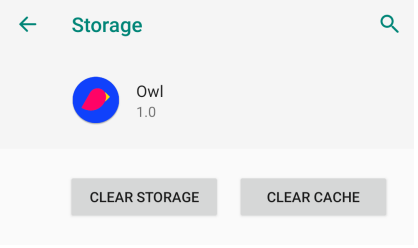
Métodos de almacenamiento.

Almacenamiento local

Se entiende por almacenamiento local a toda cosa que se aloja dentro del dispositivo. Existen distintas opciones para persistir datos. La solución a elegir dependerá de las necesidades del negocio. Por ejemplo, si se necesita que los datos sean privados para la aplicación o públicos para otras aplicaciones, también la cantidad de datos a guardar, etc.

Almacenamiento Interno

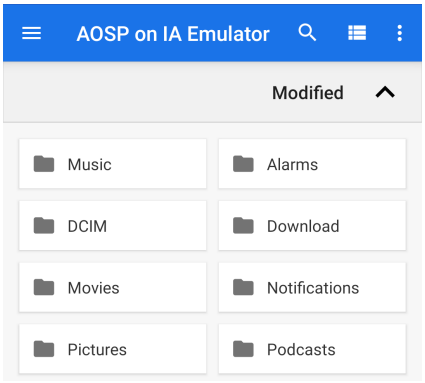
Los archivos se alojan en un directorio específico del sistema en donde el usuario no puede acceder al mismo a menos que sea root. A su vez este directorio también es privado para otras aplicaciones, por lo tanto es un lugar seguro para almacenar datos.



Este directorio es creado por el sistema al momento de instalar la aplicación, y si el usuario decide desinstalarla, los archivos que se encuentran aquí se removerán y no podrán ser recuperados. También pueden ser eliminados si el usuario accede al sector de almacenamiento de la aplicación y selecciona manualmente la opción de borrar los datos.

En este directorio también se almacena la memoria caché. Se recomienda tener control acerca del espacio utilizado para que el sistema operativo no lo borre si necesita espacio.

Almacenamiento Externo.



Los archivos se alojan en un directorio externo al del sistema, en donde el usuario tiene acceso de lectura y escritura a esta ruta, como por ejemplo las fotos, las descargas, y los videos entre otras cosas. Por lo tanto, el usuario puede modificar y eliminar los archivos que se almacenen desde la aplicación en estas ubicaciones. Como ventaja tiene que si la aplicación es desinstalada, o sus datos son borrados, los archivos almacenados aquí no se perderán.

Como es una ubicación general en donde el usuario y otras aplicaciones pueden tener acceso, para escribir y leer en estos directorios se deben solicitar permisos en tiempo de ejecución, es decir, cuando la aplicación se encuentra corriendo. Acerca de este último tema, se verá en un curso más avanzado.

Shared preferences.

Las “Preferencias compartidas” se utilizan para almacenar datos primitivos en pares de clave-valor. Si no se requiere guardar mucha información, este método de almacenamiento es una alternativa. Por defecto Android trae una API que permite leer y escribir las Shared Preferences. Estas son escritas en un archivo XML en la memoria interna del celular. Esto quiere decir que si el usuario la desinstala, o borra los datos de la app, este XML también lo hará.

¿Cómo obtenerlas?

Para acceder a las Shared Preference se debe poseer un context, como por ejemplo, la Activity. Desde aquí se podrá llamar a dos métodos dependiendo lo que se necesite realizar:

1. getSharedPreferences(nombre, modo): se utiliza si se deben almacenar múltiples shared preferences que serán identificadas por un nombre. Este nombre, que es un String, debe ser pasado como primer parámetro. El segundo parámetro es el modo en que se abrirá. Por defecto se utilizará “MODE_PRIVATE”. Ej:

```
SharedPreferences prefs = getApplicationContext().getSharedPreferences( name: "USERNAME", MODE_PRIVATE);
```

2. getPreferences(modo): se utiliza si se debe utilizar una sola shared preference. No se necesita proporcionar un nombre para identificarla. Ej:

```
SharedPreferences prefs = getPreferences(MODE_PRIVATE);
```

Si la Shared Preference no existe no es necesario crearla, ya que internamente si no existe la crea, y si existe, la obtiene para su utilización.

Una vez obtenida la instancia de la Shared Preference precisada se podrán utilizar los siguientes métodos para obtener los valores:

- getBoolean(clave, valorPorDefecto)
- getFloat(clave, valorPorDefecto)
- getInt(clave, valorPorDefecto)
- getLong(clave, valorPorDefecto)
- getString(clave, valorPorDefecto)

El parámetro “clave” será el nombre con el que se almacenará, y el valor por defecto funcionará en caso de que previamente no se haya guardado algún dato para evitar que devuelva null. Ej:

```
SharedPreferences prefs = getPreferences(MODE_PRIVATE);
prefs.getString( key: "USERNAME",  defValue: "");
prefs.getInt( key: "AGE",  defValue: 0);
```

¿Cómo agregar/modificar los valores?

Para agregar o modificar los valores primero se debe obtener la instancia de la SharedPreference. Luego se debe obtener una instancia "editable" de la misma. Para conseguir esto se generará una instancia del tipo "SharedPreferences.Editor" que será creada a partir de la instancia de SharedPreference obtenida aplicándole el método "edit()".

Una vez obtenida la instancia, se podrán utilizar los siguientes métodos:

- putBoolean(clave, valor)
- putFloat(clave, valor)
- putInt(clave, valor)
- putLong(clave, valor)
- putString(clave, valor)

Luego de realizar los cambios, se debe llamar a un método específico para guardar las preferencias. Actualmente se pueden utilizar los métodos commit() ó apply(). La diferencia radica en que el commit hará que se almacenen los datos de manera sincrónica (es decir, el thread de la vista), y el apply lo hará de manera asincrónica. Ej:

```
SharedPreferences prefs = getPreferences(MODE_PRIVATE);
SharedPreferences.Editor editor = prefs.edit();

editor.putString("USERNAME", "Romina");
editor.putInt("AGE", 33);
editor.apply();
```

Buena práctica de uso

Se recomienda que tanto el parámetro del nombre de la SharedPreference como el de la clave de cada dato se guarde como una constante y se use desde allí. Es una buena práctica ya que luego no existirán problemas de nombres. Si en un lugar se escribe con una letra mayúscula y en otro con una en minúscula, los archivos y/o las claves van a diferir.

Base de datos.

SQLite

Android nativamente nos brinda la posibilidad de crear y manipular las bases de datos mediante la clase SQLiteOpenHelper. Dependiendo la dificultad de consultas que debamos realizar hacia la base de datos, se recomienda utilizar un ORM, ya que la utilización del SQLiteOpenHelper implica desarrollar las consultas a bajo nivel.

ORMLite

Es un ORM muy liviano, potente y completo. Utilizándolo nos abstraeremos de tener que escribir a mano las consultas tales como SELECT * FROM TABLA WHERE… Link de la página: http://ormlite.com/sqlite_java_android_orm.shtml

Para empezar a utilizarlo debemos agregar las siguientes librerías en las dependencias del gradle de la app:

```
implementation 'com.j256.ormlite:ormlite-core:5.1'
implementation 'com.j256.ormlite:ormlite-android:5.1'
```

Una vez incluidas las librerías y sincronizado el proyecto, debemos crear una clase que permita comunicarnos con la base de datos. Esta clase debe extender a OrmLiteSqliteOpenHelper. Luego debemos implementar dos métodos que son obligatorios tenerlos y generar un constructor específico para instanciar nuestra base de datos.

```
public class DBHelper extends OrmLiteSqliteOpenHelper {

    private static final String NOMBRE_DB = "DB_EXAMENES";
    private static final int VERSION_DB = 1;

    public DBHelper(Context context) { super(context, NOMBRE_DB, factory: null, VERSION_DB); }

    @Override
    public void onCreate(SQLiteDatabase database, ConnectionSource connectionSource) {

    }

    @Override
    public void onUpgrade(SQLiteDatabase database, ConnectionSource connectionSource, int oldVersion, int newVersion) {

    }
}
```

En este caso estamos manejando una base de datos llamada “DB_EXAMENES”. Cada vez que instanciemos la clase DBHelper, vamos obtenerla, y para instanciarla sólo debemos pasarle el Context.

El método onCreate() se ejecutará sólo una vez para crear la base de datos, y aquí es donde debemos generar las tablas.

En el método onUpgrade() deberemos validar las versiones de la base de datos. Por ejemplo, si agregamos una nueva tabla, debemos sumarle “uno” a la versión y en este método realizar la acción de agregar esta tabla nueva sólo si la versión es la anterior más uno.

Una vez finalizado el esqueleto de la clase DBHelper deberemos modificar la clase Examen . Para realizar la vinculación de campos con variables de la clase, ORMLite utiliza annotations. Para definir el nombre de la tabla debemos hacerlo mediante @DatabaseTable, y para el nombre de un campo, mediante @DatabaseField.

```
@DatabaseTable(tableName = "Examenes")
public class Examen {

    @DatabaseField(id = true)
    private Integer id;
    @DatabaseField
    private String materia;
    @DatabaseField
    private String fecha;

    public Examen(){

    }
```

Una vez creada la clase POJO junto con las anotaciones de tabla/campos, tenemos que volver a la clase DBHelper para la creación de la tabla Libros dentro de el método onCreate():

```
@Override
public void onCreate(SQLiteDatabase database, ConnectionSource connectionSource) {
    try {
        TableUtils.createTable(connectionSource, Examen.class);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Si poseemos más de una tabla, debemos repetir los mismos pasos que hicimos con la clase Examen y agregarla debajo de la línea de la creación de la tabla.

Lo que queda pendiente ahora es crear una clase que permita comunicarnos con esta tabla. Siguiendo el ejemplo de los laboratorios, sería nuestra clase singleton ExamenManager. Luego, desde el activity que debe acceder a la información, consultamos la misma a través de ella. Lo que haría que en vez de guardar/pedir la lista de examenes en memoria, se haría mediante la base de datos.

```
public class ExamenManager {

    private static  ExamenManager instance;

    public static ExamenManager getInstance(){
        if (instance == null){
            instance = new ExamenManager();
        }
        return instance;
    }

    public ExamenManager() {
    }
}
```

Para realizar esto, en la clase ExamenManager, debemos crear una instancia Dao<Clase1, Clase2>. El primer parámetro es la clase con la que se operará (Examen), y el segundo es la clase que pertenece el Id, en este caso sería Integer, por lo tanto la definición de la instancia sería: Dao<Examen, Integer>.

```
private static  ExamenManager instance;
Dao<Examen, Integer> dao;
```

Ademas tenemos que tener un constructor que inicialice la conexión con la base de datos. Para esto usaremos la clase OpenHelperManager indicándole a nuestro objeto “dao” la clase con la que va a trabajar.

```
public ExamenManager(Context context) {
    OrmLiteSqliteOpenHelper helper = OpenHelperManager.getHelper(context, DBHelper.class);
    try {
        dao = helper.getDao(Examen.class);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

***Nota**: se agregó el Context como parámetro ya que se requiere para la conexión con la base de datos. Por lo tanto debemos agregarlo también en el método del “getInstance”.*

Teniendo nuestro objeto dao inicializado podemos realizar las consultas a la base de datos, como por ejemplo un alta, baja, consulta, etc. Ejemplo:

```
public List<Examen> getExamenes() throws Exception {
    return dao.queryForAll();
}

public void agregarExamen(Examen examen) throws Exception {
    dao.create(examen);
}
```

La llamada a estos métodos será obteniendo la instancia del ExamenManager primero y luego le pedimos los exámenes, por ejemplo..

Ejemplos: http://ormlite.com/android/examples/

Descargar base de datos

Si necesitamos descargar la base de datos que creamos, debemos:

1. Tener el emulador encendido (o el dispositivo conectado).

2. Acceder desde el Android Studio a “Device File Explorer” (ubicado en el margen inferior derecho).

3. Ingresar a la carpeta “data”.

4. Ingresar a la subcarpeta “data”.

5. Buscar el nombre del paquete de nuestra aplicación y acceder.

6. Ingresar a la subcarpeta “databases”.

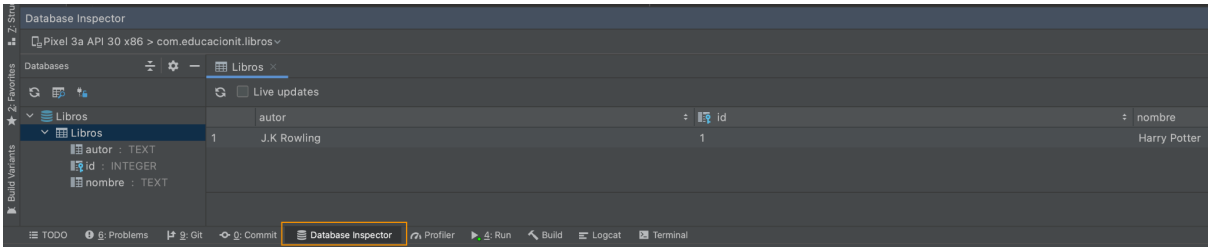
7. Buscar la creada y darle click derecho.

8. Seleccionar la acción deseada.

Inspector de base de datos

El Android Studio trae una herramienta que permite inspeccionar la base de datos en profundidad. Se pueden realizar consultas de todo tipo a las tablas.

Para acceder a esta herramienta se deberá presionar sobre el tab “Database Inspector” que se encuentra en el tab inferior izquierdo del IDE.



Práctica 4. A

El objetivo de este laboratorio es agregar un Icono “+” al ítem del menú que permite navegar desde la Lista de Exámenes hacia la Pantalla Agregar Examen.

Práctica 4. B

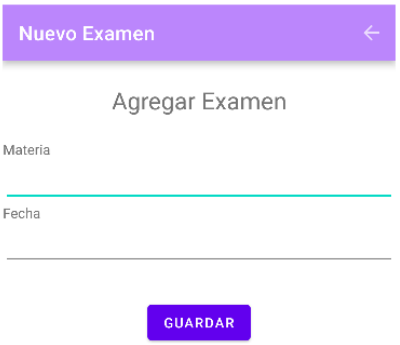
El objetivo de este laboratorio es aplicar los conceptos vistos acerca de las Shared Preferences.

A partir del estado del Checkbox (tildado/destildado) - Recordar Usuario en la pantalla de Inicio de sesión, se deberá guardar el nombre de usuario / contraseña si se encuentra en TRUE la primera vez.

A su vez, si al entrar a la aplicación se encuentran los valores guardados, se deberá ingresar directamente al MainActivity y saludar al Usuario con su Nombre en un Toast.

Práctica 4. C

El objetivo de este laboratorio es maquetar la vista de Agregar Exámen con su lógica aplicando los conceptos vistos de la Base de Datos. Incluir también el Toolbar y un botón de menú que nos permita volver a la pantalla principal.



Pantalla Agregar Examen

- **LinearLayout**: orientación vertical y margin de 8dp
- **TextView**: título de agregar usuario con textSize de 24sp, margin 24dp, gravity = center.
- **TextView**: “Materia”
- **EditText**: id = etMateria
- **TextView**: “Fecha”
- **EditText**: id = etFecha
- **Button**: id = btnGuardar, layout_gravity = center_horizontal, margin 32dp

Práctica 4. D

El objetivo de este laboratorio es modificar la activity Agregar Examen para agregar su lógica aplicando los conceptos vistos de la Base de Datos.

Pasos a realizar:

1. Agregar librerías del ORMLite en el gradle.
2. Crear clase DBHelper extendiendo a OrmLiteSqliteOpenHelper. Definir nombre y versión de DB. Agregar constructor e implementar métodos.
3. Modificar la clase Examen con las anotaciones correspondientes.
4. Crear la clase ExamenManager para que contenga la conexión y los métodos de llamada a la base de datos. Tener en cuenta que hay que agregarle el Context tanto en el constructor como en el método getInstance.
5. Modificar el getExamenes() para que devuelva datos de la BDD. Agregar el funcionamiento del botón guardar. Utilizar try/catch por si no se pudo realizar la transacción.