PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

2º DAM

TRATAMIENTO DE DATOS EN ANDROID

CONTENIDOS DE LA UNIDAD

ALMACENAMIENTO DE DATOS
AGENDA DE CONTACTOS
SISTEMA INTERNO DE FICHEROS
ALMACENAMIENTO DE DATOS EXTERNO
BASES DE DATOS

Almacenamiento de datos. La clase SharedPreferences

La clase SharedPreferences o también conocida como preferencias, no son más que datos que una aplicación debe guardar, para personalizar la experiencia del usuario.

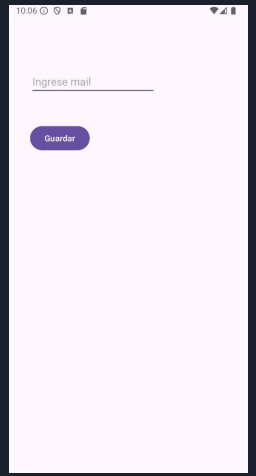
Como por ejemplo: información personal, opciones de presentación y configuración.

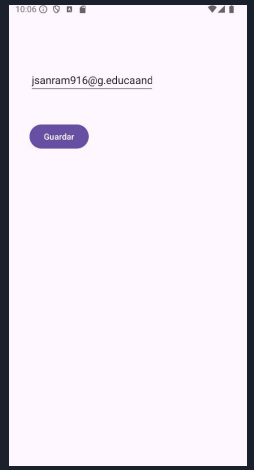
Android proporciona distintas opciones para el almacenamiento permanente de datos , es decir, que los datos ingresados no se pierdan cuando se cierra la aplicación.

Cuando tenemos que almacenar una cantidad limitada de datos, es adecuado utilizar la clase SharedPreferences. Por ejemplo, configuraciones de la aplicación como pueden ser colores de pantalla, nivel actual en un juego, datos iniciales de controles de entrada de datos, etc.

Almacenamiento de datos. La clase SharedPreferences

Realizaremos un ejemplo en el que introduciremos, en un campo de texto, un email. Se cerrará la aplicación, y al volver a abrirla debe permanecer el correo introducido.
E igualmente, si volvemos a introducir otro email, o lo borramos, y cerramos la aplicación, ésta almacenará el último valor.





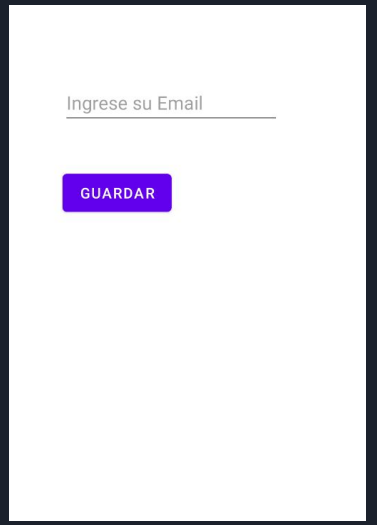
Almacenamiento de datos. La clase SharedPreferences

- Crearemos un nuevo proyecto, llamado SharedPreferences, tal y como hemos hecho en otras ocasiones.
  Seleccionaremos una *Empty views activity* y eliminaremos el Hello World.
- Añadiremos un widget E-mail
- Añadiremos un botón
- Establecemos un id txt_mail para el campo de texto
- Establecemos las distancias entre botones y bordes
- Modificamos el tamaño del texto si lo necesitamos

Almacenamiento de datos. La clase SharedPreferences

- Añadimos el contenido necesario en strings.xml

```
<resources>
  <string name="app_name">SharedPreferences</string>
  <string name="txt_mail">Ingrese su Email</string>
  <string name="Button">Guardar</string>
</resources>
```



Almacenamiento de datos. La clase SharedPreferences

- “Conectamos” parte gráfica y lógica en el Main

```
import ...

public class MainActivity extends AppCompatActivity {

  private EditText et1;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     et1 = findViewById(R.id.txt_mail);

  }
```

Almacenamiento de datos. La clase SharedPreferences

- Ahora crearemos un método para poder recuperar esas preferencias que queremos guardar. Para ello creamos un objeto de tipo SharedPreferences y lo recuperaremos con get.
- El primer parámetro será “datos” y el segundo Context.MODE_PRIVATE
- Escribiremos en nuestro método onCreate():

```
SharedPreferences preferences = getSharedPreferences("datos", Context.MODE_PRIVATE);
et1.setText(preferences.getString("mail", ""));
```

Obtenemos una instancia de SharedPreferences con getSharedPreferences. Esto busca el archivo de preferencias llamado "datos". Si no existe, Android lo crea automáticamente.
MODE_PRIVATE indica que el archivo de preferencias solo será accesible por la aplicación que lo creó.
Luego, con preferences.getString("mail", ""), obtenemos el valor asociado con la clave "mail". Si no hay ningún valor guardado, se devuelve un string vacío ("").

https://developer.android.com/training/data-storage/shared-preferences?hl=es-419

Almacenamiento de datos. La clase SharedPreferences

- Y crearemos el método para guardar la información:

```
public void guardar(View view){
  SharedPreferences preferencias = getSharedPreferences("datos", Context.MODE_PRIVATE);
  SharedPreferences.Editor obj_editor = preferencias.edit();
  obj_editor.putString("mail", et1.getText().toString());
  obj_editor.commit();
  finish();
}
```

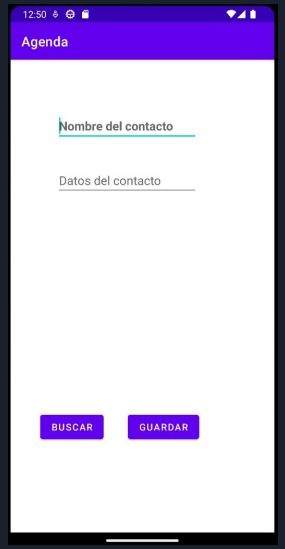
En el método *guardar*, de nuevo obtenemos las preferencias y creamos un objeto *SharedPreferences.Editor* para modificar el archivo de preferencias.
Con *putString()*, guardamos el valor del campo de texto asociado a la clave "mail".
Finalmente, hacemos *commit()* para aplicar los cambios y *finish()* para cerrar la actividad.

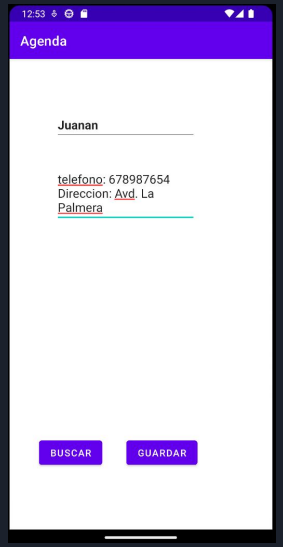
https://developer.android.com/training/data-storage/shared-preferences?hl=es-419

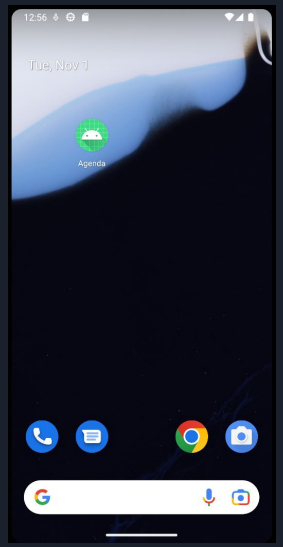
Almacenamiento de datos.
Práctica: agenda de contactos

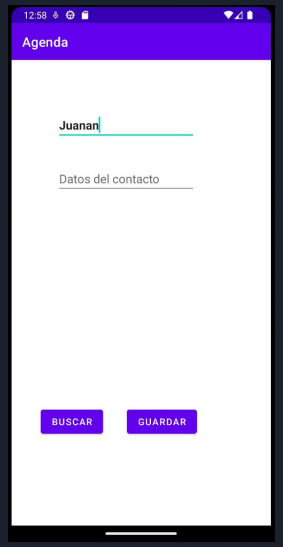
Almacenamiento de datos. Agenda de contactos

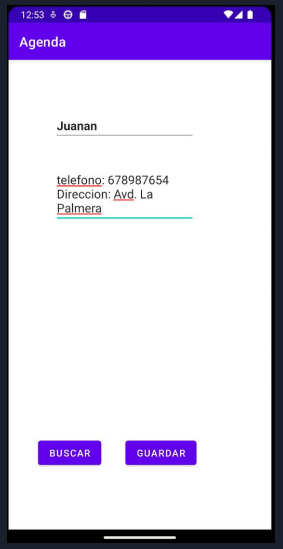
La idea es crear una aplicación donde podamos almacenar datos con SharedPreferences, y tras cerrar la app, podamos buscar y recuperar esos datos.











Almacenamiento de datos. Agenda de contactos

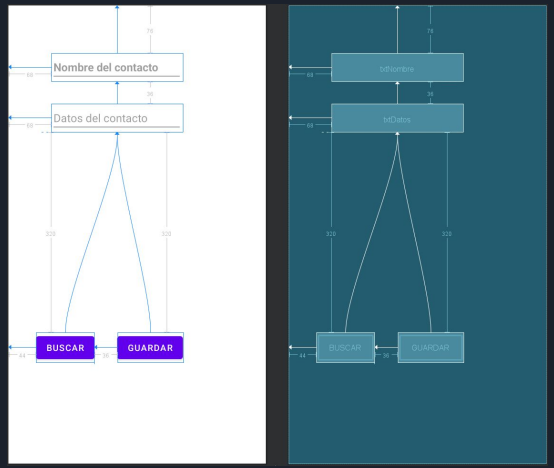
Haciendo uso de SharedPreferences podremos almacenar datos en nuestra agenda, pero de forma limitada, ya que dicha funcionalidad no está diseñada para almacenar gran cantidad de datos. Es más para pequeños parámetros de configuración o preferencias.

💡 ¿Dónde se almacenan las preferencias compartidas?

Almacenamiento de datos. Agenda de contactos

- Crearemos un proyecto llamado Agenda
- Seleccionamos una activity vacía
- Añadimos dos Edit Text:
  - Plain Text (id: txtNombre)
  - Multiline Text (id: txtDatos)
- Añadimos dos botones (serán “Guardar” y “Buscar”)
- Establecemos las distancias entre los controles desde blueprint
- Ponemos el texto de Name en negrita (textStyle)
- Creamos las referencias desde strings.xml

```
<resources>
  <string name="app_name">Agenda</string>
  <string name="txt_nombre">Nombre del contacto</string>
  <string name="txt_datos">Datos del contacto</string>
  <string name="botonGuardar">Guardar</string>
  <string name="botonBuscar">Buscar</string>
</resources>
```



Almacenamiento de datos. Agenda de contactos

- Agregamos las @referencias
- Creamos los objetos
  - `private EditText etNombre, etDatos;`
- Relacionamos parte gráfica y lógica
  - `etNombre = findViewById(R.id.txtNombre);`
  - `etDatos = findViewById(R.id.txtDatos);`
- Creamos el método para el botón guardar

```
public void guardar(View view){
  String nombre = etNombre.getText().toString();
  String datos = etDatos.getText().toString();

  SharedPreferences preferencias = getSharedPreferences("agenda", Context.MODE_PRIVATE);
  SharedPreferences.Editor obj_editor = preferencias.edit();
  obj_editor.putString(nombre, datos);
  obj_editor.commit();

  Toast.makeText(this, "El contacto ha sido guardado", Toast.LENGTH_SHORT).show();
}
```

Almacenamiento de datos. Agenda de contactos

Las preferencias se almacenan en el sistema de archivos. El modo define quién tiene acceso a las preferencias:

- MODE_PRIVATE es el modo operativo de las preferencias. Es el modo predeterminado y significa que solo la aplicación que realiza la llamada accederá al archivo creado.
- MODE_WORLD_READABLE, otras aplicaciones pueden leer el archivo creado, pero no pueden modificarlo.
- MODE_WORLD_WRITEABLE, otras aplicaciones también tienen permisos de escritura para el archivo creado.

Almacenamiento de datos. Agenda de contactos

- Creamos el método para el botón buscar

```
public void buscar(View view){
  String nombre = etNombre.getText().toString();

  SharedPreferences preferencias = getSharedPreferences("agenda", Context.MODE_PRIVATE);
  String datos = preferencias.getString(nombre, "");

  if(datos.length() == 0){
     Toast.makeText(this, "No se encontro ningún registro", Toast.LENGTH_SHORT).show();
  } else {
     etDatos.setText(datos);
  }
}
```

Almacenamiento de datos. Agenda de contactos

La estructura condicional nos servirá para comprobar si el contacto buscado existe, y en tal caso, muestre los datos.
En primer lugar comprobamos si la cadena es vacía “” quiere decir que no ha encontrado el contacto. En caso contrario, colocamos en etDatos los datos de nuestro objeto SharedPreferences.

- De nuevo nos dirigimos a la parte gráfica y establecemos los onClick de los botones.

Almacenamiento de datos.
Sistema interno de ficheros

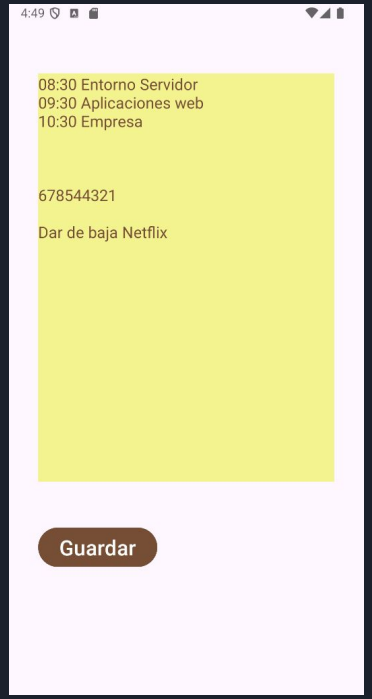
Otra alternativa de almacenar datos en nuestro dispositivo Android, es el empleo de un archivo de texto o fichero que se guardará en el almacenamiento interno del dispositivo móvil.

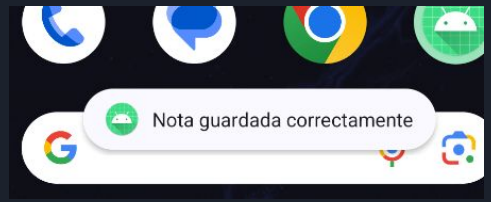
Los ficheros almacenados sólo son accesibles para la aplicación que los creó, no pueden ser leídos por otras aplicaciones, ni siquiera por el usuario del teléfono o dispositivo Android.

Cada aplicación dispone de una carpeta especial para almacenar ficheros:
(/data/data/nombre_del_paquete/files).

La ventaja de utilizar esta carpeta, es que cuando se desinstala la aplicación los ficheros que hemos creado se eliminarán.

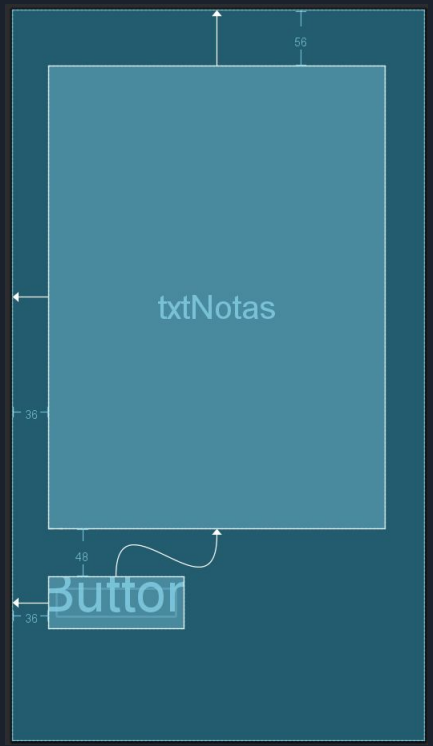
Almacenamiento de datos.
Sistema interno de ficheros





Almacenamiento de datos. Sistema interno de ficheros

- Crearemos un proyecto llamado BlocDeNotas
- Añadimos un Multiline (id: txtNotas)
- Añadimos un botón
- Establecemos las distancias



Almacenamiento de datos. Sistema interno de ficheros

- Editamos el diseño de nuestro Multiline
  - Pondremos el atributo Gravity en [start|top]
  - textColor en #FFFFFF
  - background en #FF9800
  - Aumentamos un poco el tamaño del texto del botón
- Modificamos nuestro strings.xml

```
<resources>
  <string name="app_name">BlocDeNotas</string>
  <string name="boton">Guardar</string>
</resources>
```

- Cambiamos las @referencias

Almacenamiento de datos. Sistema interno de ficheros

- Crearemos los atributos
  - `private EditText et1;`
- Y conectamos con la parte gráfica
  - `et1 = findViewById(R.id.txtNotas);`
- Al trabajar ahora con ficheros de texto, tenemos que “encontrarlos” con ayuda de fileList(), el cual devuelve un array con los ficheros almacenados por nuestra app. También, crearemos un espacio temporal en memoria para almacenar dichos ficheros.
  - `String archivos [] = fileList();`

Almacenamiento de datos. Sistema interno de ficheros

```
if(archivoExiste(archivos, "blocdenotas.txt")){
   try {
       InputStreamReader archivo = new InputStreamReader(openFileInput("blocdenotas.txt"));
       BufferedReader br = new BufferedReader(archivo);
       String linea = br.readLine();
       String notaCompleta = "";

       while(linea != null){
          notaCompleta = notaCompleta + linea + "\n";
          linea = br.readLine();
       }
       br.close();
       archivo.close();
       et1.setText(notaCompleta);
   }catch (IOException e){

   }
}
```

Si efectivamente existe nuestro fichero “blocdenotas.txt” en la lista de todos los ficheros, *luego declararemos ese método*, le indicamos que vamos a crear un objeto de tipo InputStreamReader (para poder leerlo).
Una vez abierto el archivo, lo leemos con BufferReader. Leemos línea a línea dicho archivo con br.readLine(). Con un while rellenamos el string notaCompleta. Cerramos la lectura de nuestro archivo y establecemos en la variable et1 todo lo leído en este método.

Almacenamiento de datos. Sistema interno de ficheros

Creamos el método archivoExiste:

```
private boolean archivoExiste(String archivos[], String nombreArchivo){
  boolean res = false;
  for(int i = 0; i < archivos.length; i++){
     if(nombreArchivo.equals(archivos[i])){
        res=true;
     }
  }
  return res;
}
```

Y el método para el botón guardar:

```
public void guardar(View view){
  try {
     OutputStreamWriter archivo = new OutputStreamWriter(openFileOutput("blocdenotas.txt", Activity.MODE_PRIVATE));
     archivo.write(et1.getText().toString());
     archivo.flush();
     archivo.close();
  }catch (IOException e){

  }
  Toast.makeText(this, "Nota guardada correctamente", Toast.LENGTH_SHORT).show();
  finish();
}
```

Almacenamiento de datos. BASE DE DATOS

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto, y almacenados sistemáticamente para su posterior uso.
Existen programas denominados sistemas gestores de bases de datos (SGBD), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

En Android, podremos encontrar una herramienta poderosa para el almacenamiento y consulta mediante bases de datos, conocida como: SQLite.

Almacenamiento de datos. BASE DE DATOS

SQLite. es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y sobre todo ser de código libre.

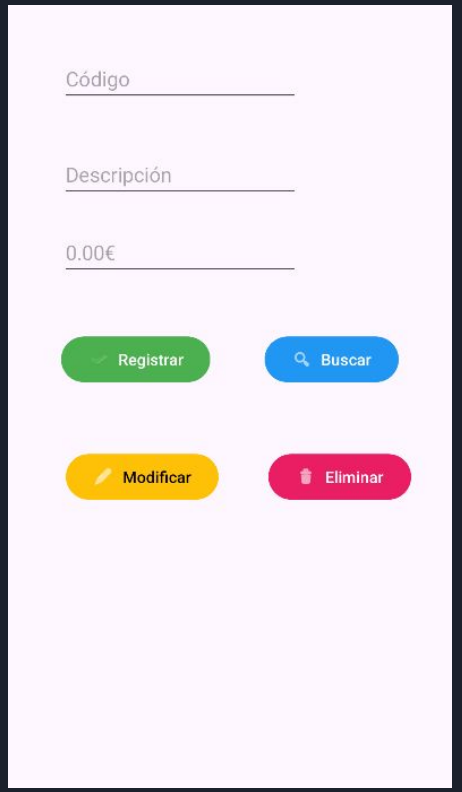
En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite será a través de una clase auxiliar llamada SQLiteOpenHelper.

La clase SQLiteOpenHelper tiene tan sólo un constructor, que normalmente no necesitaremos sobrescribir, y dos métodos abstractos, onCreate() y onUpgrade(), que deberemos personalizar con el código necesario para crear nuestra base de datos y para actualizar su estructura respectivamente.

Almacenamiento de datos. BASE DE DATOS

Realizaremos una app que nos permita almacenar productos.

- Para ello necesitaremos introducir un código, que no podrá estar vacío y solamente puede ser numérico. En caso de error deberá lanzar un toast. También deberemos añadir una descripción y un precio.
- También podremos buscar ese producto introduciendo su código.
- Una vez buscado, podremos modificarlo.
- Y de la misma manera, eliminarlo.



Almacenamiento de datos. BASE DE DATOS

Crearemos un nuevo proyecto llamado ProductosSQLite

- Seleccionaremos una activity vacía
- Creamos una nueva clase haciendo click derecho en com.example.XXX.basededatos, es decir, la carpeta que alberga la parte lógica de nuestra aplicación. Esa nueva clase se llamará AdminSQLiteOpenHelper.

Esta clase tendrá como utilidad administrar la base de datos que vamos a crear.

- Importamos: `import android.database.sqlite.SQLiteOpenHelper ;`
- Usamos la herencia para extender dicha clase a SQLiteOpenHelper
- Para solucionar el error que nos aparece, sobreescribimos los métodos onCreate y onUpgrade.
- A continuación, creamos el constructor de 4 parámetros que nos aconseja Android Studio.

```
public AdminSQLiteOpenHelper(Context context, String name,
SQLiteDatabase.CursorFactory factory, int version) {
    super(context, name, factory, version);
}
```

Almacenamiento de datos. BASE DE DATOS

Una vez creado el *esqueleto* de nuestra clase para administrar la base de datos, nos vamos al método onCreate y alteramos el parámetro a nuestro gusto: así se llamará nuestra base de datos:

```
public void onCreate(SQLiteDatabase baseDeDatos) {
```

A continuación, dentro, creamos la tabla *articulos* que almacenará nuestros artículos. Contendrá las columnas y claves primarias que se muestran:

```
baseDeDatos.execSQL("create table articulos(codigo int primary key, descripcion text,
precio real)");
```

Almacenamiento de datos. BASE DE DATOS

Esta clase extiende SQLiteOpenHelper y es responsable de gestionar la creación y actualización de la base de datos.

Método onCreate
Se ejecuta al crear la base de datos por primera vez.
Define una tabla llamada *articulos* con tres columnas: *codigo* (clave primaria), *descripcion* (texto), y *precio* (número real).

Método onUpgrade
Aunque en esta implementación está vacío, normalmente se usa para actualizar la estructura de la base de datos entre versiones.

Constructor AdminSQLiteOpenHelper
Configura la base de datos con los parámetros dados por el contexto de la aplicación, el nombre, el cursor y la versión.

Esta estructura permite realizar las operaciones CRUD (crear, leer, actualizar y eliminar) para gestionar los productos en la base de datos SQLite.

Almacenamiento de datos. BASE DE DATOS

Ahora sí, volvemos a nuestra vista diseño y añadimos:

- un editText de tipo Number (id: txt_codigo)
- un editText de tipo Plain Text (id: txt_descripcion)
- un editText de tipo Number Decimal (id: txt_precio)

Tras esto agregamos cuatro botones y asignamos las distancias desde la vista blueprint.
Creamos los strings:

```
<resources>
    <string name="app_name">BaseDeDatos</string>
    <string name="txt_codigo">Código del producto</string>
    <string name="txt_descripcion">Descripción del producto</string>
    <string name="txt_precio">Precio del producto</string>
    <string name="BotonRegistrar">Registrar Producto</string>
    <string name="BotonBuscar">Buscar Producto</string>
    <string name="BotonEliminar">Eliminar Producto</string>
    <string name="BotonModificar">Modificar Producto</string>
</resources>
```

Almacenamiento de datos. BASE DE DATOS

- Volvemos a la parte lógica de nuestra aplicación y creamos los atributos editText: `private EditText et_codigo, et_descripcion, et_precio;`
- Dentro de onCreate creamos la relación gráfica/lógica
- Método para dar de alta los productos:

```
public void registrar(View view){
    AdminSQLiteOpenHelper admin =  new AdminSQLiteOpenHelper( this, "administracion", null, 1);
    SQLiteDatabase baseDeDatos = admin.getWritableDatabase()  ;

    String codigo =  et_codigo.getText().toString() ;
    String descripcion =  et_descripcion.getText().toString() ;
    String precio =  et_precio.getText().toString() ;

    if(!codigo.isEmpty() && !descripcion.isEmpty() && !precio.isEmpty()){
        ContentValues registro =  new ContentValues() ;

        registro.put( "codigo", codigo);
        registro.put( "descripcion", descripcion) ;
        registro.put( "precio", precio) ;

        baseDeDatos.insert( "articulos", null, registro) ;

        et_codigo.setText("");
        et_descripcion.setText("");
        et_precio.setText("");

        Toast.makeText(this,"Registro exitoso", Toast.LENGTH_SHORT).show() ;
    } else{
        Toast.makeText(this, "Debe llenar todos los campos" , Toast.LENGTH_SHORT).show() ;
    }
baseDeDatos.close() ;
}
```

Almacenamiento de datos. BASE DE DATOS

- Método para consultar productos:

```
public void buscar(View view){
    AdminSQLiteOpenHelper admin =new AdminSQLiteOpenHelper(this, "administracion", null, 1);
    SQLiteDatabase baseDeDatabase = admin.getWritableDatabase()

    String codigo = et_codigo.getText().toString();

    if(!codigo.isEmpty()){
        Cursor fila = baseDeDatabase.rawQuery
                ("select descripcion, precio from articulos where codigo =" codigo, null);

        if(fila.moveToFirst()){
            et_descripcion.setText(fila.getString(0));
            et_precio.setText(fila.getString(1));

        } else {
            Toast.makeText(this,"No existe el artículo", Toast.LENGTH_SHORT).show();
        }

    } else {
        Toast.makeText(this, "Debes introducir el código del artículo," Toast.LENGTH_SHORT).show();
    }
        baseDeDatabase.close();
}
```

Almacenamiento de datos. BASE DE DATOS

- Método para eliminar productos:

```
public void eliminar(View view){
    AdminSQLiteOpenHelper admin =new AdminSQLiteOpenHelper(this, "administracion", null, 1);
    SQLiteDatabase baseDatabase = admin.getWritableDatabase()

    String codigo = et_codigo.getText().toString();

    if(!codigo.isEmpty()){

        int cantidad = baseDatabase.delete("articulos", "codigo=" + codigo, null);

        et_codigo.setText("");
        et_descripcion.setText("");
        et_precio.setText("");

        if(cantidad == 1){
            Toast.makeText(this, "Artículo eliminado exitosamente", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "El artículo no existe", Toast.LENGTH_SHORT).show();
        }

    } else {
        Toast.makeText(this, "Debes de introducir el código del artículo", Toast.LENGTH_SHORT).show();
    }
        baseDatabase.close();
}
```

Almacenamiento de datos. BASE DE DATOS

- Método para modificar productos:

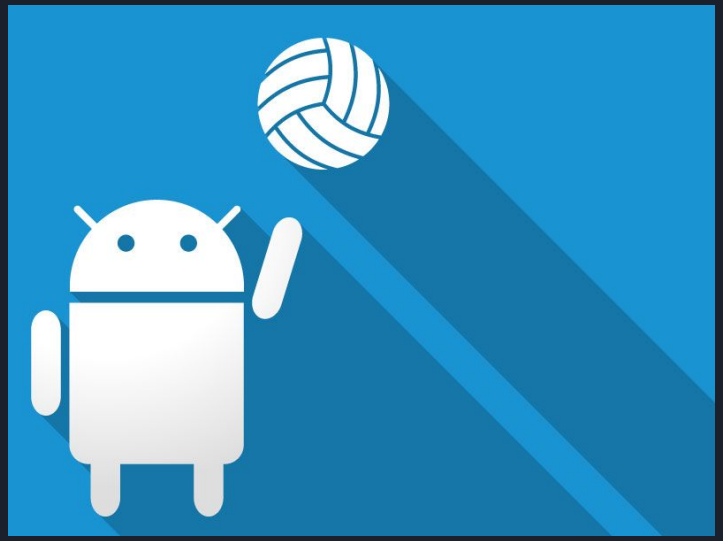
```
public void modificar(View view){
    AdminSQLiteOpenHelper admin =new AdminSQLiteOpenHelper(this, "administracion", null, 1);
    SQLiteDatabase baseDatabase = admin.getWritableDatabase();

    String codigo = et_codigo.getText().toString();
    String descripcion = et_descripcion.getText().toString();
    String precio = et_precio.getText().toString();

    if(!codigo.isEmpty() && !descripcion.isEmpty() && !precio.isEmpty()){

        ContentValues registro =new ContentValues();
        registro.put("codigo", codigo);
        registro.put("descripcion", descripcion);
        registro.put("precio", precio);

        int cantidad = baseDatabase.update("articulos", registro, "codigo=" + codigo, null);

        if(cantidad == 1){
            Toast.makeText(this, "Artículo modificado correctamente", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "El artículo no existe", Toast.LENGTH_SHORT).show();
        }

    } else {
        Toast.makeText(this, "Debe llenar todos los campos", Toast.LENGTH_SHORT).show();
    }
   baseDatabase.close();
}
```

Almacenamiento de datos. BASE DE DATOS

- Ponemos los métodos creados a nuestros botones, y listo.
- Buscamos información en internet sobre dónde se almacena dicha base de datos.

Almacenamiento de datos. BASE DE DATOS | VOLLEY

Ahora, aprenderemos a conectar una App a una base de datos MySQL, sea en local o en remoto. Para ello usaremos la librería Volley de Android.
https://google.github.io/volley/

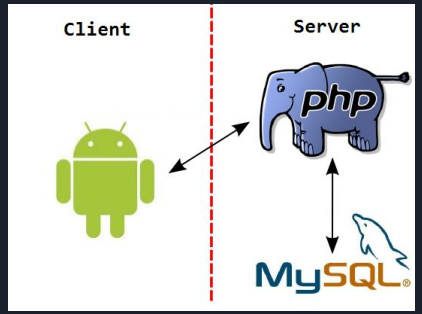


Almacenamiento de datos. BASE DE DATOS | VOLLEY

La principal ventaja de conectar nuestra app a una base de datos MySQL y no a SQLite, es que los datos permanecen fuera de nuestro dispositivo, ya sea en localhost en nuestro PC (por ejemplo con xampp y PhpMyadmin) o en cualquier hosting que nos ofrezca una base de datos.

La complejidad de esta separación datos-lógica será los puertos y URL necesarias para que los datos viajen por internet.

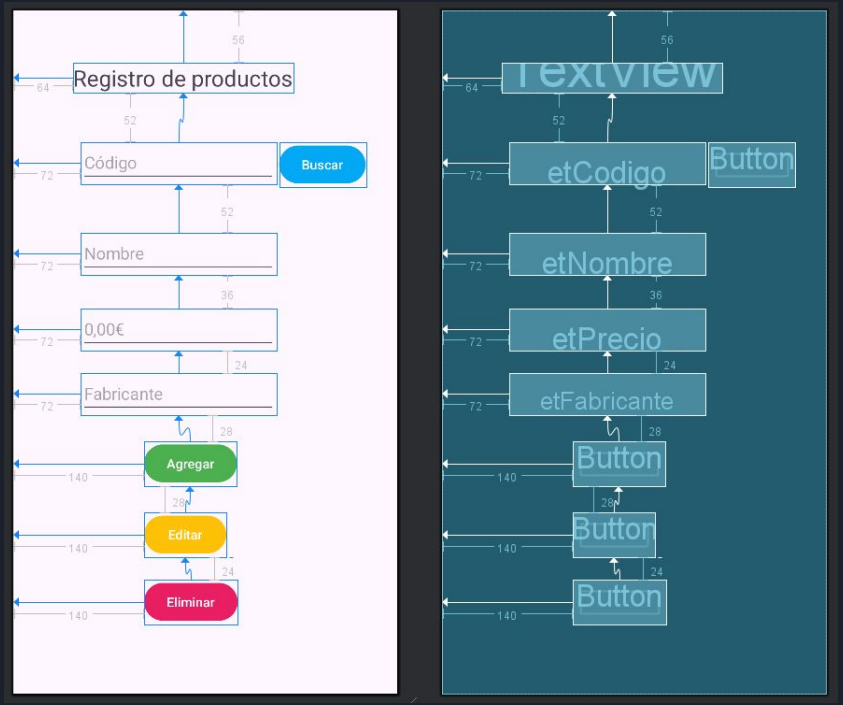
Realizaremos una práctica de un CRUD de producto.



Almacenamiento de datos. BASE DE DATOS | VOLLEY

Queremos crear una app donde registrar productos y también poder buscarlos, editarlos y eliminarlos. Y, claro, que todos esos datos sean persistentes en una BD.

Crearemos la parte gráfica estableciendo las distancias, haciendo uso de strings y estableciendo las ids.



Almacenamiento de datos. BASE DE DATOS | VOLLEY

También tendremos que crear una base de datos óptima para almacenar dichos datos. Le pondremos el nombre que deseemos, por ejemplo *registro_producto* y creamos una tabla, llamada, por ejemplo, *producto*.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**codigo** 🔑	varchar(4)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	2	**nombre**	varchar(50)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	3	**precio**	decimal(6,2)			No	*Ninguna*			Cambiar Eliminar Más
☐	4	**fabricante**	varchar(25)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más

Almacenamiento de datos. BASE DE DATOS | VOLLEY

En el archivo AndroidManifest.xml añadiremos el permiso:

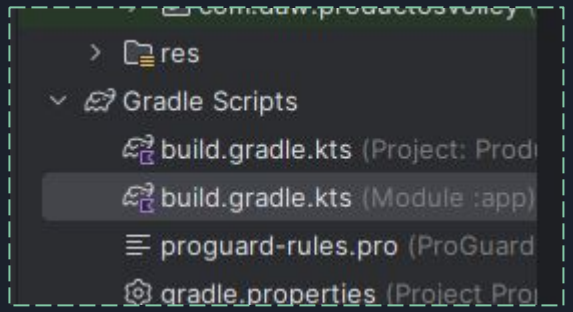
```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

y también:

```
android:usesCleartextTraffic="true"
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

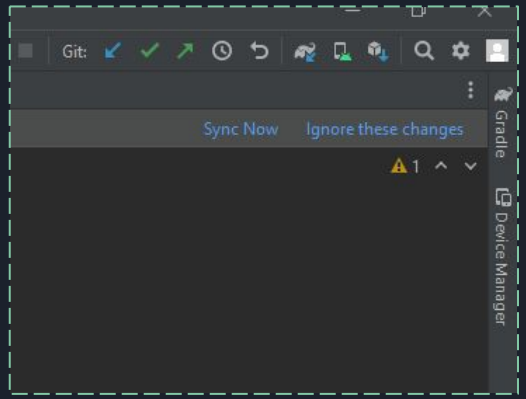
Ahora llega el momento de importar/añadir la librería Volley. Para ello nos dirigimos al archivo build.gradle.



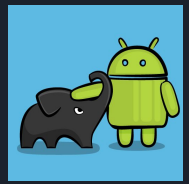
Añadimos la dependencia

```
implementation("com.android.volley:volley:1.2.1" )
```

Y sincronizamos para que se complete el proceso:



¿Qué es Gradle?
https://openwebinars.net/blog/que-es-gradle/



Almacenamiento de datos. BASE DE DATOS | VOLLEY

En la parte lógica de nuestra app crearemos la relación lógica/gráfica y además añadiremos un nuevo atributo para las consultas SQL:

```
RequestQueue requestQueue;
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

Para esta práctica, y como es como es habitual a la hora de conectar varias capas: android, mysql, datos, conexiones, lógica, etc. Hay que hacer uso de los microservicios. Con esto ayudamos a nuestra app a que no tenga que saber qué forma tiene nuestra Base de datos. Independiza las capas.

Los microservicios que usaremos para nuestra app tendrán forma de archivo php y serán los siguientes:

- conexion.php
- insertar_producto.php
- buscar_producto.php
- editar_producto.php
- eliminar_producto.php

Almacenamiento de datos. BASE DE DATOS | VOLLEY

Recordemos que el lenguaje PHP es un lenguaje interpretado en el lado del servidor, con lo que debemos almacenarlos en uno. En nuestro caso, podemos usar xampp.
Así que en la carpeta htdocs de xampp, crearemos una nueva carpeta llamada registrar_producto, por ejemplo. Y ahí dentro crearemos esos 5 archivos.

En el microservicio conexion.php establecemos la conexión enlazando $variables con las columnas de nuestra base de datos. Donde pondremos nombre, dirección, contraseña de nuestra BD.

Almacenamiento de datos. BASE DE DATOS | VOLLEY

conexion.php

```php
<?php

$hostname= 'localhost';
$database= 'registro_producto';
$username= 'root';
$password= '';

$conexion= new mysqli($hostname, $username, $password, $database);
if ($conexion->connect_errno) {
        echo "lo sentimos, error al conectar";
}

?>
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

insertar_producto.php

```php
<?php

include 'conexion.php';
$codigo=$_POST['codigo'];
$nombre=$_POST['nombre'];
$precio=$_POST['precio'];
$fabricante=$_POST['fabricante'];

$consulta="insert into producto values('".$codigo."', '".$nombre."','".$precio."','".$fabricante."')";
mysqli_query($conexion, $consulta) or die(mysqli_error());
mysqli_close($conexion);

?>
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

buscar_producto.php

```php
<?php

include 'conexion.php';
$codigo = $_GET['codigo'];

$consulta = "SELECT * FROM producto WHERE codigo = '$codigo'";
$resultado = $conexion->query($consulta);

while ($fila = $resultado->fetch_assoc()) {
  $producto[] = array_map('utf8_encode', $fila);
}

echo json_encode($producto);
$resultado->close();

?>
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

editar_producto.php

```php
<?php

include 'conexion.php';
$codigo=$_POST['codigo'];
$nombre=$_POST['nombre'];
$precio=$_POST['precio'];
$fabricante=$_POST['fabricante'];

$consulta="update producto set nombre = '".$nombre."', precio='".$precio."', fabricante='".$fabricante."' where codigo = '".$codigo."'";
mysqli_query($conexion, $consulta) or die(mysqli_error());
mysqli_close($conexion);

?>
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

eliminar_producto.php

```php
<?php

include 'conexion.php';
$codigo=$_POST['codigo'];

$consulta="delete from producto where codigo = '".$codigo."'";
mysqli_query($conexion, $consulta) or die(mysqli_error());
mysqli_close($conexion);

?>
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

Una vez listos nuestros microservicios, crearemos métodos en MainActivity.java para hacer uso de ellos a través de los datos que se recogen del formulario gráfico de nuestra app.

Crearemos los siguientes métodos:

- `public void insertarProducto(View view)`
- `public void buscarProducto(View view)`
- `public void editarProducto(View view)`
- `public void eliminarProducto(View view)`
- `private void limpiarFormulario()`

Almacenamiento de datos. BASE DE DATOS | VOLLEY

```
public void insertarProducto(View view){
    StringRequest stringRequest = new StringRequest(Request.Method.POST, "http://10.0.2.2/registrar_producto/insertar_producto.php", new
Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        Toast.makeText(getApplicationContext(), "OPERACION EXITOSA", Toast.LENGTH_SHORT).show();
    }
  }, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
    }
  }){
    @Nullable
    @Override
    protected Map<String, String> getParams() throws AuthFailureError {
        Map<String, String> parametros=new HashMap<String, String>();
        //ESTOS SON LAS $VARIABLES DEL MICROSERVICIO EN PHP
        parametros.put("codigo", edtCodigo.getText().toString());
        parametros.put("nombre", edtProducto.getText().toString());
        parametros.put("precio", edtPrecio.getText().toString());
        parametros.put("fabricante", edtFabricante.getText().toString());

        return parametros;
    }
  };
  requestQueue = Volley.newRequestQueue(this);
  requestQueue.add(stringRequest);
}
```

Almacenamiento de datos. BASE DE DATOS | VOLLEY

```
public void buscarProducto(View view){
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest("http://10.0.2.2/registrar_producto/buscar_producto.php?codigo="
+edtCodigo.getText().toString() , new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            JSONObject jsonObject = null;
            for (int i = 0; i < response.length(); i++) {
                try {
                    jsonObject = response.getJSONObject(i);
                    edtNombre.setText(jsonObject.getString("nombre"));
                    edtPrecio.setText(jsonObject.getString("precio"));
                    edtFabricante.setText(jsonObject.getString("fabricante"));
                } catch (JSONException e) {
                    Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHORT).show();
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast.makeText(getApplicationContext(), "ERROR DE CONEXIÓN", Toast.LENGTH_SHORT).show();
        }
    }
    );
    requestQueue=Volley.newRequestQueue(this);
    requestQueue.add(jsonArrayRequest);
}
```