INSTITUTO TECNOLÓGICO SUPERIOR DE LA REGIÓN DE LOS LLANOS

INGENIERIA INFORMÁTICA.

8 “U” INFORMÁTICA.

MATERIA

DESARROLLO DE APP MÓVILES 2

Unidad 2

Practica1

Agenda (datos básicos)

DOCENTE:

MGTI. Juan Carlos Morales Ponce

Alumno:

Carlos Emilio Córdova Lechuga.

No. De Control: 17030081

Guadalupe Victoria, Dgo. A 22 de abril de 2021

TECNOLÓGICO NACIONAL DE MÉXICO

Descripción

En el presente reporte se explicará la realización de la aplicación Agenda (**Datos básicos)**, esta práctica se pretende hacer una agenda, donde se guardarán datos como: imagen del Usuario, nombre, carrera y teléfono.

Todos estos datos serán guardados en base de datos remota. Estos datos serán usados en la misma aplicación, pero en otro apartado, donde se podrá escoger el usuario y se presentaran lo datos de este.

Esta práctica destaca por usar La librería de Volley, es una biblioteca HTTP que facilita y agiliza el uso de redes en apps para Android. Ofrece programación automática de solicitudes de red.

En términos generales es lo que hará la aplicación, esta cuenta con tres actividades: una para escoger la operación de guardar o visualizar la agenda, otras dos actividades donde se realizan las operaciones dichas.

La explicación será de forma como el usuario hará uso de la aplicación.

Descargar la librería Volly

En el “Module: app” de los Gradle Scripts se agrega una línea de código la cual hará la descargar las librerías Volley automáticamente (Imagen 1).

```
//Incluir la dependencias volley
implementation 'com.android.volley:volley:1.1.0'
```

(imagen 1)

Las librerías serán usadas en las actividades de guardar y visualizar los datos de la agenda.

Manifest

En este apartado lo que se realizo fue crear el permiso de internet, esto para que la aplicación puede ser uso de este, como se dijo anteriormente esta utilizara bases de datos remotas que está alojadas en un servidor (AwardSpace). También el permiso de lectura, ya que las imágenes que se guardarán serán obtenidas del explorador de archivos local (Imagen 2).

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.agenda_u2_app">

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

(imagen 2)

Se agrego la siguiente propiedad en la etiqueta application, networkSecurityConfig la cual establece parámetros de seguridad desde un archivo XML (Imagen 3).

```
<application
    android:networkSecurityConfig="@xml/network_security"
    android:allowBackup="true"
```

(imagen 3)

En el archivo XML lo que se crea es la Configuración de seguridad de red, en este caso se permite el tráfico de red de texto sin cifrar, para todas las comunicaciones de red desde este proceso. Este se creó en la siguiente dirección res>xml>network_security.xml (Imagen 4).

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="true"/>
</network-security-config>
```

(imagen 4)

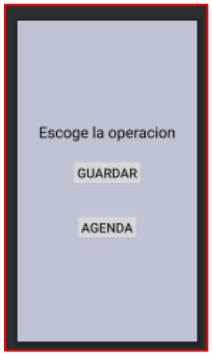
1. Actividad Principal

La actividad principal solo cuenta con dos botones, los cuales tienen como propósito dirigir a las otras actividades de guardar y visualizar los datos de la agenda.

- **Activity.xml**

Para la interfaz de la actividad principal como se comentó se cuenta con dos botones:

En la codificación de la interfaz podemos encontrar un LinearLayaut principal el cual tiene una orientación Vertical donde todos los objetos se agreguen uno debajo de otro (Imagen 1.1).




```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#86B9583"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

(imagen 1.1)

Para la parte de los botones estos cuentan con atributos similares, como el tamaño, un tamaño de texto y un margen superior o separado en cuanto a los componentes que están en la parte superior (Imagen 1.2).

```
<Button
    android:id="@+id/guardar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="40dp"
    android:text="Guardar"
    android:textSize="30dp" />
```

(imagen 1.2)

```
<Button
    android:id="@+id/agenda"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="65dp"
    android:text="Agenda"
    android:textSize="30dp" />
```

Esto es todo por parte del diseño de la actividad principal

- **Activity.java**

Para la parte lógica, se creó los componentes que serán utilizados en la actividad, en este caso los botones y enseguida se estos se enlazan con los componentes del xml (Imagen 1.3).

```
public class MainActivity extends AppCompatActivity {
    Button guardar, agenda;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        guardar = findViewById(R.id.guardar);
        agenda = findViewById(R.id.agenda);
```

(imagen 1.3)

Se crearon lo métodos para cada uno de los botones, al dar clic a cualquier botón lo que hace es iniciar una nueva instancia a la que se va dirigir pasando una Intent a startActivity, actividades tales como **Guaradr_adenda y agenda_visor** (Imagen 1.4)

```
guardar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent ir_A = new Intent( packageContext: MainActivity.this, guardar_Agenda.class);
        startActivity(ir_A);
    }
});
agenda.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent ir_a = new Intent( packageContext: MainActivity.this, agenda_visor.class);
        startActivity(ir_a);
    }
});
```

(imagen 1.4)

Actividad guardar_agenda

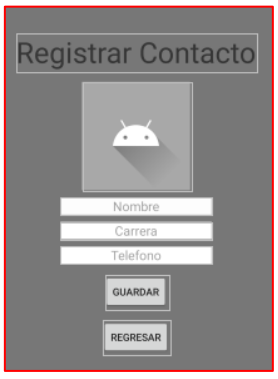
Esta actividad hace uso de funciones volley, como se comentó al inicio Volley es una biblioteca HTTP que facilita y agiliza el uso de redes en apps para Android.

En esta actividad lo que se hace es la inserción de los datos a la base de datos remota, por eso es importante la librería volley. Los datos que se tomaran, es una imagen que será codificada a texto, esto gracias a un método, el nombre de la persona, la carrera y su teléfono.

- **guardar_agenda.xml**

En la parte de la interfaz cuenta con componentes como (Imagen 1.5):

- **ImageView.**
- **TextView.**
- **Boton.**
- **EditText.**



(imagen 1.5)

En la codificación de la interfaz podemos encontrar un LinearLayaut principal el cual tiene una orientación Vertical donde todos los objetos se agreguen uno debajo de otro, contendrá atributos como color de fondo, orientación, etc. (Imagen 1.6).

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#777777"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".guardar_Agenda">
```

(imagen 1.6)

Enseguida un ImageView el cual tiene atributos como tamaño de largo y ancho especifico y una imagen que aparecerá por default (Imagen 1.7).

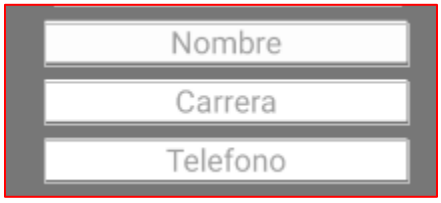
```
<ImageView
    android:id="@+id/img"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:background="#5BFFFFFF"
    app:srcCompat="@drawable/ic_launcher_foreground" />
```

(imagen 1.7)

```
<EditText
    android:id="@+id/nombre"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:background="#FDFDFD"
    android:ems="10"
    android:hint="Nombre"
    android:textAlignment="center"/>
<EditText
    android:id="@+id/carrera"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:background="#FFFFFF"
    android:ems="10"
    android:hint="Carrera"
    android:textAlignment="center"/>
<EditText
    android:id="@+id/telefono"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:background="#FFFFFF"
    android:ems="10"
    android:hint="Telefono"
    android:textAlignment="center" />
```

Se utilizaron tres EditText que tienen como propósito que el usuario teclee sus datos para registrar (Nombre, Carrera y Teléfono), los 3 tienen los mismos atributos, un tamaño especifico tanto de texto como largo y ancho, un atributo hint que muestra un texto por default y un tipo de letra (Imagen 1.8).

(imagen 1.8)

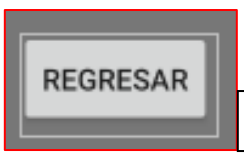


Para terminar, se utilizaron un par de botones los cual tienen propósitos diferente, uno ejecuta el método para para guardar los valores en la base de datos, el otro para cambiar a la actividad principal, estos tienen atributos como tamaño especifico, un texto y una separación del objeto de arriba (Imagen 1.8 y 1.9).

```
<Button
    android:id="@+id/guardar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Guardar"
    android:layout_marginTop="15dp"/>
```

(imagen 1.8)

```
<Button
    android:id="@+id/regresar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Regresar"
    android:layout_marginTop="15dp"/>
```



(imagen 1.9)

- **guardar_agenda.xml**

Ahora bien, para empezar, se declararon todos los componentes, variables de tipo String, de tipo int estática que será utilizada en un método que será explicado más adelante (Imagen 1.10).

```
public class guardar_Agenda extends AppCompatActivity {
    private static final int REQUEST_SELECT_PHOTO = 1  ;
    Button guardar;
    EditText nombre, carrera, telefono;
    ImageView img;
    Bitmap myBitmap = null;
    String pathFile;
    String imgString;
```

(imagen 1.10)

Se crea variables (**PathFile**) para guardar la ruta de la imagen, un **bitmap** el cual es la estructura donde se almacenan los pixeles que conforman un gráfico, en este caso las imágenes que se redimensionaran, una variable de tipo String (**imgString**), la cual almacenara la codificaron de la imagen a texto (Imagen 1.10).

Se crearon objetos que se encargarán de la petición de las solicitudes y la manipulación la petición (Imagen 1.11).

```
RequestQueue requestQueue;
StringRequest stringRequest;
```

(imagen 1.11)

Una variable de tipo String que contendrá la URL de conexión al servidor en internet, que será utilizada como parámetro cuando se inicializa la petición (Imagen 1.12).

```
String url = "http://fasesceclphp.atwebpages.com/";
```

(imagen 1.12)

Dentro del onCreate solo se enlazan los componentes en el archivo **XML** (Imagen 1.13)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_guardar__agenda);
    guardar = findViewById(R.id.guardar);
    nombre = findViewById(R.id.nombre);
    carrera = findViewById(R.id.carrera);
    telefono = findViewById(R.id.telefono);
    img = findViewById(R.id.img);
```

(imagen 1.13)

Para iniciar se tratará de explicar el proceso paso a paso de como extraer la ruta de la imagen. Bien al pulsar la imagen (img) se hace al llamado al método LlamarPortada (Imagen 1.14).

(imagen 1.14)

```
img.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        llamarExplorador();
    }
});
```

Este método utiliza **Intent** de tipo **ACTION_GET_CONTENT** al que le hemos indicado como tipo imagen. Esto hará que Android nos lance un diálogo de selección de qué fuente queremos la foto, también le pasamos un entero que hemos definido al principio, **ACTIVITY_SELECT_IMAGE**, que nos ayudará a recoger luego el resultado usando el método **onActivityResult** (imagen 1.15).

```
private void llamarExplorador() {
    android.content.Intent ii = new Intent(android.content.Intent.ACTION_PICK);
    ii.setDataAndType(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,MediaStore.Images.Media.CONTENT_TYPE);
    startActivityForResult(ii, REQUEST_SELECT_PHOTO);
}
```

(imagen 1.15)

Ya obtenida la ruta de la imagen desde la aplicación externa. Para esto usamos el método **onActivityResult** (). Hacemos uso del **Switch** que según sea el caso se ejecutara el código correspondiente, en este caso se hizo el llamado del caso **REQUEST_SELECT_PHOTO** el cual (PD. Estos códigos ya están establecidos para hacer la operación) imagen 2.16.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode) {
        case REQUEST_SELECT_PHOTO:
            if( resultCode != 0 ) {
                Cursor c = managedQuery(data.getData(), projection: null, selection: null, selectionArgs: null, sortOrder: null);
                if( c.moveToFirst() ) {
                    int i =0;
                    myBitmap = null;
                    while (myBitmap==null) {
                        pathFile = c.getString(i);
                        myBitmap = redimensionarImagen(pathFile);
                        img.setImageBitmap(myBitmap);
                        i++;
                    }
                }
            }
        default: break;
    }
```

(imagen 1.16)

Al ejecutar el código se declara una variable de tipo **int (i)** igualada a cero y la variable Bitmap declarada como global se iguala a **null** dentro del **while** se adquiere la ruta de la imagen y es guardada en su respectiva variable (**pathFile**), después se utiliza un método **Redimensionarimagen** que este le da la densidad de pixeles, de una forma que los reduce (Imagen 1.17).

```
public Bitmap redimensionarImagen(String absolutePath){
    BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(absolutePath, options);
    int imageWidth = options.outWidth;
    int imageHeight = options.outHeight;
    int ratio;
    options.inJustDecodeBounds = false;
    if (imageWidth >imageHeight) {
        ratio = imageWidth/360;
    }
    else {
        ratio = imageHeight/360;
    }
    options.inSampleSize = ratio;
    return BitmapFactory.decodeFile(absolutePath, options);
}
```

(imagen 1.17)

Entonces este resultado se gurda en el bitmap, este como dijimos se utiliza para devolver un mapa de bits (img) para después establecer un mapa de bits como

contenido de **ImageView**, esto para que el usuario vea la imagen que extrajo del móvil y verifique que es la que escogió (Imagen 1.16).

Al finalizar esto se manda llamar al método **BitMapToString**, el cual se le pasará como parámetro el Bitmap obtenido (Imagen 1.18).

```
    }
    imgString = BitMapToString(myBitmap);
}
public String BitMapToString(Bitmap bitmap){
    ByteArrayOutputStream baos=new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.PNG, quality: 100, baos);
    byte [] b=baos.toByteArray();
    String temp= Base64.encodeToString(b, Base64.DEFAULT);
    return temp;
}
```

(imagen 1.18)

Este método tiene como objetivo convertir el Bitmap a texto y regresarlo para posteriormente guardarlo en la variable imgString, que posteriormente será utilizada a la hora de la inserción (Imagen 1.18).

Ahora sí, es momento de explicar el método Guardar, este se ejecuta al pulsar el botón Guardar.

Lo primero que se realiza es inicialización el objeto que realizará la petición al servidor (requestQueue), la función Volley lo que hace es proporcionar un método de conveniencia Volley.newRequestQueue, que configura el RequestQueue por nosotros mediante valores predeterminados y, luego, inicia la cola (Imagen 1.19).

```
//Se inicializa el objeto que realizará la petición al servidor
requestQueue = Volley.newRequestQueue( context: guardar_Agenda.this);
```

(imagen 1.19)

Enseguida se estructura e inicializa la petición mediante el método POST, con los parámetros de conexión URL (variable que tiene guardado la URL de conexión) (Imagen 1.20)

```
//Se estructura e inicializa la petición con los parámetros de conexión y la información a gestionar
stringRequest  = new StringRequest(Request.Method.POST,  url: url + "registross.php", new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
    }
}){
```

(imagen 1.20)

Para agregar la información que será enviada al servidor (al archivo PHP) hacemos uso del Hashmap, lo que es una estructura matricial en la cual se agrega información que será enviada al servidor (al archivo PHP) Imagen 1.21.

```java
@Override
protected HashMap<String, String> getParams() {
    HashMap<String, String> map = new HashMap<>();
    map.put("nombre", nombre.getText().toString());
    map.put("carrera", carrera.getText().toString());
    map.put("telefono", telefono.getText().toString());
    map.put("string_img", imgString);
    return map;
}
```

(imagen 1.21)

Se utilizó el método **put** el cual almacena el valor especificado y lo asocia con la clave especificada en este mapa

Los parámetros que utiliza este método son dos, clave y valor, donde clave es el argumento de la izquierda que se identifica en el php y valor, en este caso son los que agrega el usuario en los edittext que quiere registrar el nombre, carrera y teléfono, en el caso de la imagen se agregara la variable que contiene el que se convirtió a string, ya al final solo se regresa (Imagen 1.21).

Ya para finalizar solo se agrega el stringRequest a la cola de peticiones (Imagen 1.22).

```java
};
requestQueue.add(stringRequest);
```

(imagen 1.22)

El guardado de datos se hace las veces que el usuario requiera.

Código completo:

```java
guardar.setOnClickListener((v) -> {
        requestQueue = Volley.newRequestQueue( context: guardar_Agenda.this);
        stringRequest  = new StringRequest(Request.Method.POST,  url: url + "registross.php", new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
            }
        }){
            @Override
            protected HashMap<String, String> getParams() {
                HashMap<String, String> map = new HashMap<>();
                map.put("nombre", nombre.getText().toString());
                map.put("carrera", carrera.getText().toString());
                map.put("telefono", telefono.getText().toString());
                map.put("string_img", imgString);
                return map;
            }
        };
        requestQueue.add(stringRequest);
});
```

Por parte de **php**, quedo de la siguiente manera (Imagen 1.23):

```php
<?php
$par1=$_REQUEST['nombre'];
$par2 =$_REQUEST['carrera'];
$par3 =$_REQUEST['telefono'];
$par4 =$_REQUEST['string_img'];

$conexion = mysqli_connect("fdb30.awardspace.net", "3808179_personas", "2DznVw/Q0XWp*,yC", "3808179_personas");
$sql = "INSERT INTO agenda(nombre, carrera, telefono, img)values('$par1', '$par2'.'$par3', '$par4')";
if($conexion->query($sql)===true){
    echo"Se agregaron";
}

else{
    echo "Error: ".$sql."<br>".$conexion->error;
}
mysql_close($conexion);
?>
```

(imagen 1.22)

Los valores enviados con la clave se guardan en las variables $par, enseguida se hace la conexión con la base de datos, posterior se crea la inserción a la base de datos con los valores guardados en las variables locales que fueron enviados de la app (Imagen 1.22).

Base de datos:

+ Opciones

nombre	carrera	telefono	img
Carlos	administraciÃ³n	6761105385	iVBORw0KGgoAAAANSUhEUgAAATMAAAGUCAIAAADWI1JfAAAAA3...
Carlos	administraciÃ³n	6761105385	iVBORw0KGgoAAAANSUhEUgAAATMAAAGUCAIAAADWI1JfAAAAA3...
Carlos	administraciÃ³n	6761105385	iVBORw0KGgoAAAANSUhEUgAAAWgAAAHCCAIAAABNAMQKAAAAA3...
emilio	informatica	6761086826	iVBORw0KGgoAAAANSUhEUgAAAWgAAAHCCAIAAABNAMQKAAAAA3...
Yami	informatica	6761086826	iVBORw0KGgoAAAANSUhEUgAAAPAAAAGqCAIAAAC6VV7gAAAAA3...

Ya solo para volver a la actividad principal se ejecuta el método del botón regresar, lo que hace es iniciar una nueva instancia a la que se va dirigir pasando una Intent a startActivity() imagen 1.23.

```java
regresar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent regresar = new Intent( packageContext: guardar_Agenda.this, MainActivity.class);
        startActivity(regresar);
    }
});
```

(imagen 1.21)

Esto sería todo por parte de la actividad de registro, enseguida se explicará la actividad donde se consultará los registros a través del valor seleccionado en el Spinner.

2. Actividad visor_agenda

Esta actividad vuelve hacer uso de funciones volley. Aquí se pretende que el usuario pueda visualizar los contactos y sus datos personales como Imagen, nombre, teléfono y carreara que guardo en la actividad anterior.

- **visor_agenda.xml**

Los componentes que se utilizaron en la interfaz, son los siguientes (Imagen 1.22):

- **Imageview**
- **TextView**
- **EditText**
- **Spinner**
- **Botón**

(imagen 2.1)

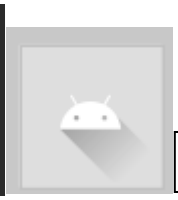
En la codificación de la interfaz podemos encontrar un LinearLayaut principal el cual tiene una orientación Vertical donde todos los objetos se agreguen uno debajo de otro, un color de fondo personalizado, un Gravity al centro esto para que todos los elementos se agreguen al centro (Imagen 2.2).

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#C8C8C8"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".agenda_visor">
```

(imagen 2.2)

Para el componente ImageView la cual al iniciar se da una imagen de fondo, esta contiene atributos como, tamaño de ancho y largo especifico con Layaut_Gravity centrado (Imagen 2.3).

```
<ImageView
    android:id="@+id/imagen"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#66FFFFFF"
    app:srcCompat="@drawable/ic_launcher_foreground" />
```

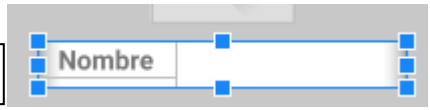


(imagen 2.3)

```
<LinearLayout
    android:layout_width="280dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="15dp"
    android:background="#FFFFFF"
    android:orientation="horizontal">
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="0.3"
        android:text="Nombre"
        android:textAlignment="center"
        android:textSize="20dp"
        android:textStyle="bold" />
    <Spinner
        android:id="@+id/spiin"
        android:layout_width="0dp"
        android:layout_height="35dp"
        android:layout_weight="0.5"
        android:textColor="#000" />
</LinearLayout>
```

Continuando con el diseño, se utilizó otro LinearLeyaut secundario este con una orientación Horizontal el cual servirá para poner los componentes uno del lado del otro. Los componentes son un **textView** “nombre” y un **spinner** en el cual aparecerán los diferentes usuarios que se encuentras registrados (Imagen 2.4).

(imagen 2.4)



Para el apartado de Carrera, se utilizó otro LinearLeyaut este con una orientación Horizontal. Los componentes que se utilizaron fueron un par de **textView**, uno para mostrar un texto predeterminado “Carrera” y el otro mostrara el resultado de la consulta por parte del capo de la carrera (Imagen 2.5).

(imagen 2.5)

```
<LinearLayout
    android:layout_width="280dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="15dp"
    android:background="#FFFFFF"
    android:orientation="horizontal">
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="0.3"
        android:text="Carrera"
        android:textAlignment="center"
        android:textSize="20dp"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/carrera"
        android:layout_width="0dp"
        android:layout_height="35dp"
        android:layout_weight="0.5"
        android:textColor="#000" />
</LinearLayout>
```

```
<LinearLayout
    android:layout_width="280dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="15dp"
    android:background="#FFFFFF"
    android:orientation="horizontal">
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="0.3"
        android:text="Telefono"
        android:textAlignment="center"
        android:textSize="20dp"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/telefono"
        android:layout_width="0dp"
        android:layout_height="35dp"
        android:layout_weight="0.5"
        android:textColor="#000" />
</LinearLayout>
```

Para el apartado de Telefono, se volvió a utilizar LinearLeyaut con una orientación Horizontal. Los componentes que se utilizaron fueron un par de **textView**, uno para mostrar un texto predeterminado “Teléfono” y el otro mostrara el resultado de la consulta por parte del capo del teléfono, (Imagen 2.6).

(imagen 2.6)

Para terminar, se utilizó un Botón, el cual se encuentra en el LinearLeyaut Principal, este componente cuenta con atributos de tamaño especifico, un texto y una separación al objeto superior (Imagen 2.7).

```
<Button
    android:id="@+id/regresar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Regresar"
    android:layout_marginTop="15dp"/>
```

(imagen 2.7)

- **visor_agenda.java**

Para iniciar con la actividad se crearon las variables y los componentes que serán usados para enlazar al xml y un arrayadapter que almacenara valores (Nombres) imagen 2.8.

```
public class agenda_visor extends AppCompatActivity {
    //Componetes del XML
    Spinner spi;
    ArrayAdapter <String> adaptador;
    TextView carrera, telefono;
    ImageView img;
    Button regresar;
```

(imagen 2.8)

Se crearon objetos que se encargarán de la petición de las solicitudes y la manipulación la petición (Imagen 2.9).

```
RequestQueue requestQueue;
StringRequest stringRequest;
```

(imagen 2.9)

Una variable de tipo String que contendrá la URL de conexión al servidor, que será utilizada como parámetro cunado se inicializa la petición (Imagen 2.10).

```
String url = "http://fasesceclphp.atwebpages.com/";
```

(imagen 2.10)

Se creo una variable (**ImageCod**) la cual servirá para guardar el dato decodificado de la imagen y un **bitmap** que es la estructura donde se almacenan los pixeles que conforman un gráfico imagen (Imagen 2.11).

```
String imagenCod;
Bitmap myBitmap = null;
```

(imagen 2.11)

Al terminar lo siguiente es enlazar los componentes al XML (Imagen 2.12)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_agenda_visor);
    spi = findViewById(R.id.spiin);
    carrera = findViewById(R.id.carrera);
    telefono = findViewById(R.id.telefono);
    img = findViewById(R.id.imagen);
    regresar = findViewById(R.id.regresar);
```

(imagen 2.12)

El en el adaptador se guardarán los diferentes nombres registrados en el base de datos, estos serán obtenidos por medio de una consulta y estos se presentarán en el Spinner (Imagen 2.13).

```
adaptador = new ArrayAdapter<>( context: agenda_visor.this, android.R.layout.simple_spinner_dropdown_item);
spi.setAdapter(adaptador);

sacar_datos();
```

(imagen 2.13)

Para consultar estos datos en el oncreate se manda llamar el método **sacar_datos** (Imagen).

En este método lo primero que se realiza es inicialización del objeto que realizará la petición al servidor, se utiliza la función Volley la cual proporciona un método de conveniencia Volley.newRequestQueue, que configura una RequestQueue por nosotros mediante valores predeterminados y, luego, inicia la cola (Imagen 2.14).

```
requestQueue = Volley.newRequestQueue( context: agenda_visor.this);
//Se estructura e inicializa la petición con los parámetros de conexión y la información a gest
stringRequest  = new StringRequest(Request.Method.POST,  url: url + "consulta.php", new Response.Listener<String>() {
```

(imagen 2.14)

Enseguida se estructura e inicializa la petición mediante el método POST, con los parámetros de conexión URL (variable que tiene guardado la URL de conexión y el php) y la información a gestionar. En el php se encarga de ejecutar la consulta, en este caso solo se requieren lo nombres (Imagen 2.15), la consulta que realiza es referente a la base de datos que esta alojada en el servidor de AwardSpace:

```
<?php
$conexion = mysqli_connect("fdb30.awardspace.net", "3808179_personas", "2DznVw/Q0XWp*,yC", "3808179_personas");
$inf = array();
$res = $conexion->query("select nombre from agenda");

if ($res->num_rows == 0) {
    $info[]=array("nombre"=>"sin info");
}

while ($fila = $res->fetch_assoc()) {
    # code...
    $info[]=array("nombre"=>$fila['nombre']);
}
$inf["datos"] = $info;
print json_encode($inf);
$conexion->close();
?>
```

(imagen 2.15)

Los valores obtenidos de la consulta se guardan en un array, que posteriormente será manipulado en Android, estos se guardan como clave-valor ejemplo: clave= “nombre” valor = “Carlos”.

Ahora bien, en Android se ejecuta el método onReponse, que es la respuesta que nos manda el servidor, en este caso el array la cual será manipulada.

Se crea un try-catch, dentro del try se obtiene la información codificada que retorno el servidor, enseguida se decodifica el objeto en un vector manipulable. Para poder recorrer todos los datos obtenidos se utilizó sentencia for que comienza mediante la declaración de la variable i y se inicializa a 0. Comprueba que i es menor al tamaño del vectorJson, si es así realiza las sentencia con éxito e incrementa i en 1 después de cada pase del bucle (Imagen 2.16).

```
    @Override
    public void onResponse(String response) {
        try {
            JSONObject objetoJSON = new JSONObject(response);
            JSONArray vectorJSON = objetoJSON.getJSONArray( name: "datos");

            for (int i = 0 ;i<vectorJSON.length();i++)
            {
                adaptador.add(vectorJSON.getJSONObject(i).getString( name: "nombre"));
            }
            adaptador.notifyDataSetChanged();
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
    }
});
requestQueue.add(stringRequest);
```

(imagen 2.16)

Dentro de este se extrae la información de la estructura dentro del vector con la clave puesta en el **php**, al ir incrementando la variable, cada dato guardado en el vector con ese número de índice se le hará una lectura y posterior se extraerá y será guardado en el arrayAdapter, hasta que la sentencia for ya no se cumpla, es decir que el vector ya no tenga más datos, ya al finalizar solo se le notifica el cambio de conjunto de datos al ArrayAdapter y estos serán agregados al spinner (Imagen 2.16).

Ya para finalizar solo se agrega el stringRequest a la cola de peticiones (Imagen 2.17).

```
};
requestQueue.add(stringRequest);
```

(imagen 2.17)

Ahora bien, como ya se tiene lo valores en el spinner, el usuario podrá seleccionar cualquier valor que este dentro de este y se hará una consulta a la base de datos con relación al valor que se escogió, esto con la finalidad de que el usuario visualice los contactos agregados.

Como en la consulta anterior se inicializa el objeto que realizará la petición al servidor, enseguida se estructura e inicializa la petición mediante el método POST, con los parámetros de conexión URL (variable que tiene guardado la URL de conexión, el php y el valor que requiere obtener el php) esta forma de enviar la llave y el valor mediante el URL no es lo recomendable, pero es una forma sencilla de poder enviárselo (Imagen 2.18).

```
spi.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        requestQueue = Volley.newRequestQueue( context: agenda_visor.this);
        stringRequest  = new StringRequest(Request.Method.POST,  url: url + "con_presonalizada.php"  +"?nombre="+ spi.getSelectedItem(),
                new Response.Listener<String>() {
```

(imagen 2.18)

También está la otra forma de enviar mediante al Hashmap, la cual al igual que como se está mandando requiere una llave y el valor, en el URL solo se le concatena la llave y el valor o datos seleccionado del spinner.

Ejemplo:

URL = http://fasesceclphp.atwebpages.com/con_presonalizada.php?nombre=”JUAN”

En el php quedo de la siguiente forma (Imagen 2.19):

```
<?php
$par1= $_REQUEST['nombre'];

$inf = array();

$conexion = mysqli_connect("fdb30.awardspace.net", "3808179_personas", "2DznVw/Q0XWp*,yC", "3808179_personas");

$res = $conexion->query("select * from agenda where nombre = '$par1'");

while ($fila = $res->fetch_assoc()) {
    # code...
    $info[]=array("nombre"=>$fila['nombre'], "carrera"=>$fila['carrera'], "telefono"=>$fila['telefono'], "img"=>$fila['img']);
}
$inf["datos"] = $info;
print json_encode($inf);
$conexion->close();
?>
```

(imagen 2.19)

Obtiene el valor mandado en el URL mediante el método request, crea la conexión a la base de datos, se hace la consulta con condición que se debe cumplir para que se devuelvan las filas, es decir selecciona todos los valores de la base de datos alojada en el servidor de AwarSpace en la tabla **agenda** con la condición de que el nombre se igual al valor de la variable que obtuvo el valor mediante el método request (Imagen 2.19).

Estos valores serán guardados en un array, que posteriormente obtenido y manipulado en el método onResponse en Android.

En Android Studio se ejecuta del método onResponce, que es la respuesta que nos manda el servidor, en este caso el array el cual contiene los valores de la consulta que posteriormente será manipulado (Imagen 2.20).

```
@Override
public void onResponse(String response) {
    try {
        JSONObject objetoJSON = new JSONObject(response);
        JSONArray vectorJSON = objetoJSON.getJSONArray( name: "datos");
        carrera.setText(vectorJSON.getJSONObject( index: 0).getString( name: "carrera"));
        telefono.setText(vectorJSON.getJSONObject( index: 0).getString( name: "telefono"));
        imagenCod = vectorJSON.getJSONObject( index: 0).getString( name: "img");
        myBitmap = StringToBitMap(imagenCod);
        img.setImageBitmap(myBitmap);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

(imagen 2.20)

Se crea un try-catch, dentro del try se obtiene la información codificada que retorno el servidor, enseguida se decodifica el objeto en un vector manipulable con la clave que se asignó en el php (Imagen 2.20).

Al tener el vector se extrae la información de la estructura dentro del vector, en este caso cada valor se establecerá en su componente, cada uno con la clave correspondiente para obtener dicha información, en el caso para establecer la imagen, en la actividad donde se guardaba los datos esta se codifico a texto, por lo que se obtendrá de esa manera, este valor se pasará como parámetro del método **StringToBitMap** que codifica el texto a un mapa de bits (Imagen 2.21).

```
public Bitmap StringToBitMap(String encodedString){
    try{
        byte [] encodeByte= Base64.decode(encodedString,Base64.DEFAULT);
        Bitmap bitmap= BitmapFactory.decodeByteArray(encodeByte,  offset: 0, encodeByte.length);
        return bitmap;
    }catch(Exception e){
        e.getMessage();
        return null;
    }
}
```

(imagen 2.21)

Al final este retorna el valor en mapa de bits o Bitmap y se establece en el imageview, con la propiedad **setImageBitmap** (Imagen 2.21)**.**

Ya para finalizar solo se agrega el stringRequest a la cola de peticiones (Imagen 2.22).

```
};
requestQueue.add(stringRequest);
```

(imagen 2.22)

Todo esto se volverá a ejecutar cada vez que el usuario escoja un valor o en este caso un contacto del spinner, lo que hará que se extraiga información de la base de datos referente a ese contacto.

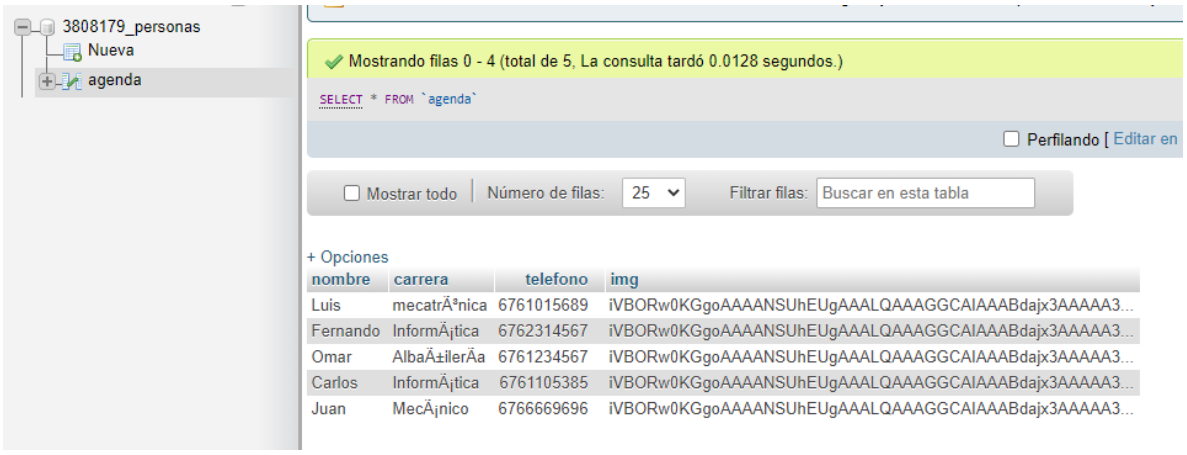
Ya solo para volver a la actividad principal se ejecuta el método del botón regresar, lo que hace es iniciar una nueva instancia a la que se va dirigir, pasando el Intent a startActivity (Imagen 2.23).

```
regresar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent regresar = new Intent( packageContext: guardar_Agenda.this, MainActivity.class);
        startActivity(regresar);
    }
});
```

(imagen 2.23)

Seria todo por explicar, espero que me diera a entender. Hay algunas cosas que se me dificulta explicar, pero intento dar lo mejor en mi explicación. Enseguida se mostrarán algunos resultados al guardar contactos y al mostrar los datos de estos.

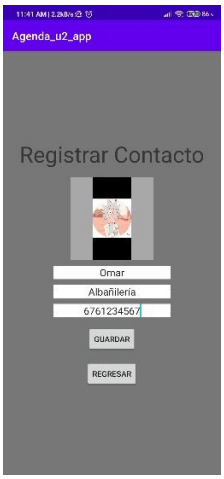
Base de datos en AwardSpace

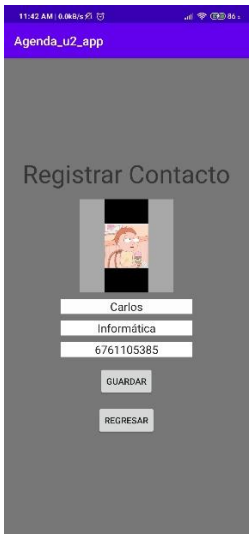


3. Resultados

Actividad principal, aquí es usuario escoge la operación que desea hacer:

Las siguientes imágenes muestran los diferentes registros que se realizaron en la actividad **guardar_agenda**.





Las siguientes imágenes muestran la selección del contacto que se registraron en la actividad **Guardar_agenda,** donde el usuario al escoger uno contacto del spinner los datos de este se muestran en los diferentes campos.