TECNOLOGICO DE ESTUDIOS SUPERIORES DE CHALCO

REPORTE DE PRACTICA:

“SYNC-ADAPTER SQLITE CON MYSQL MEDIANTE VOLLEY”

ALUMNO: JIMÉNEZ FLORES JOSE RICARDO

GRUPO: 4951

INGENIERIA EN SISTEMAS COMPUTACIONALES

MATERIA: BASE DE DATOS PARA DISPOSITIVOS MÓVILES

PROFESORA: MCC. MORALES HUERTA MARTHA GUADALUPE

DESARROLLO:

SYNC-ADAPTER SQLITE CON MYSQL MEDIANTE VOLLEY

Descripción:

Implementar la herramienta Sync-Adapter en SQLite con MySQL y librería VOLLEY donde vamos a sincronizar la base de datos SQLite de Android con el servidor . Supongamos que tenemos que enviar cierta información desde la aplicación a nuestro servidor web y que Internet no está disponible en el dispositivo en un momento determinado. Entonces, en lugar de dar error al usuario de que Internet no está disponible, podemos almacenar los datos en SQLite y enviarlos más tarde automáticamente cuando Internet esté disponible.

Procedimiento:

Se inicia el servidor XAMMP.

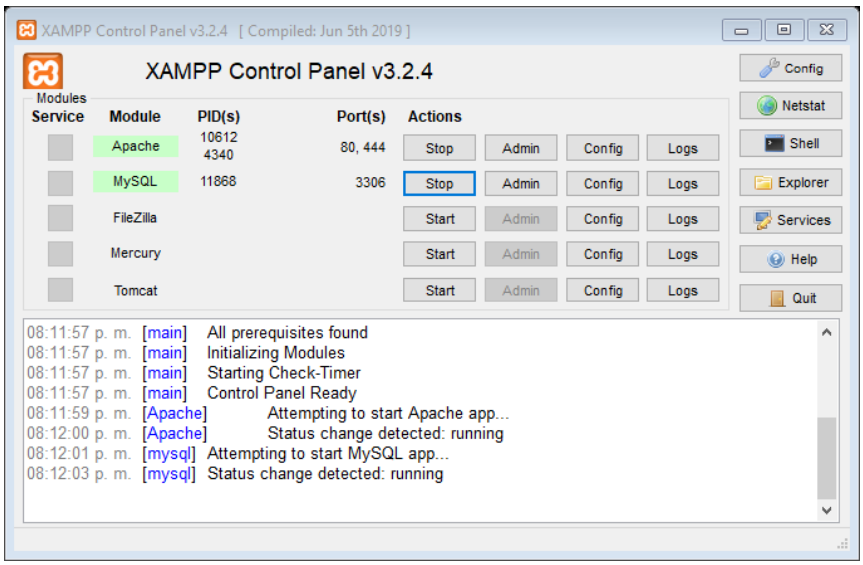

Ilustración 1. Iniciamos XAMPP.

Creamos una nueva Base de Datos llamada contactos y dentro una tabla llamada usuario con tres campos: id, nombre y teléfono como se muestra en la imagen.

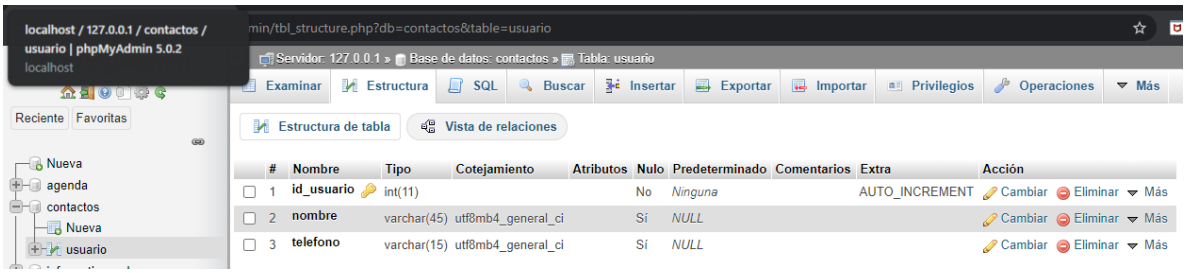

Ilustración 2. BD creada en phpMyAdmin.

CREACION DEL WEB SERVICE

Crearemos 1 fichero .php dentro de la carpeta agenda en xampp/htdocs para la conexión de la base de datos y el Web Service que va a estar proporcionando información a la aplicación “save.php” donde guardara los datos de conexión a la BD que se lleguen a ocupar en todas las clases PHP.

C:\xampp\htdocs\WebService\save.php • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
save.php

```php
<?php

define('DB_HOST','localhost');//Nombre del Host local
define('DB_USERNAME','root');//Nombre del usuario
define('DB_PASSWORD','');//Contraseña
define('DB_NAME', 'contactos');//Nombre de la BD a conectar

//Conectar con la BD
$conn = new mysqli(DB_HOST, DB_USERNAME, DB_PASSWORD, DB_NAME);

//Comprobando la conexión exitosa
if($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
    exit();
}

//Haciendo una matriz para almacenar la respuesta
$response = array();

//Si hay una solicitud de publicación, envia por el método POST
if($_SERVER['REQUEST_METHOD']=='POST'){

 //Obteniendo el nombre y teléfono de la solicitud

 //Creando una declaración para insertar en la base de datos
 $stmt = $conn->prepare("INSERT INTO usuario (nombre, telefono) VALUES (?, ?)");

 //Vinculando el parámetro a las declaraciones
 $stmt->bind_param("ss", $name, $phone);

```

Ilustración 3. Parte 1 Código save.php.

```php
$name = $_POST['name'];
$phone = $_POST['phone'];

//Si los datos se insertan correctamente
if($stmt->execute()){
//Haciendo una respuesta de éxito
$response['error'] = false;
$response['message'] = 'Register saved successfully';
}else{
//Si no da respuesta de falla
$response['error'] = true;
$response['message'] = 'Please try later';
}

}else{
 $response['error'] = true;
 $response['message'] = "Invalid request";
}

//Mostrando los datos en formato json
echo json_encode($response);
?>
```

Ilustración 4. Parte 2 Código save.php.

Creamos un proyecto nuevo en AndroidStudio “SyncAdapterMysql” como se muestra a continuación:

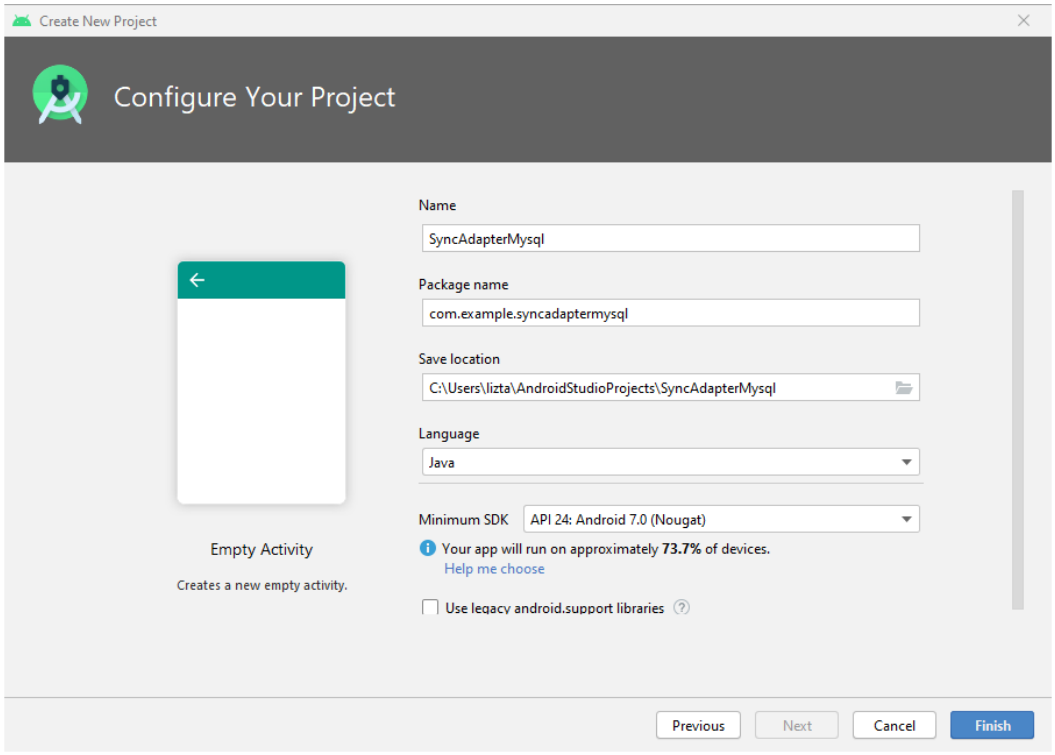

Ilustración 5. Proyecto nuevo en AndroidStudio.

Ahora otorgamos permiso de “Internet” en el archivo AndroidManifest.xml.

SQLiteDataHelper.java | MainActivity.java | NameAdapter.java | AndroidManifest.xml | Name.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.syncadaptermysql">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="SyncAdapterMysql"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="true">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".NetworkStateChecker">
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
            </intent-filter>
        </receiver>
    </application>
```

Ilustración 6. Otorgar permisos.

En el activity_main.xml colocamos dos campos: Nombre, Teléfono y un boton para guardar el registro; además de un ListView para que al momento de realizar la consulta se muestre dentro del mismo.

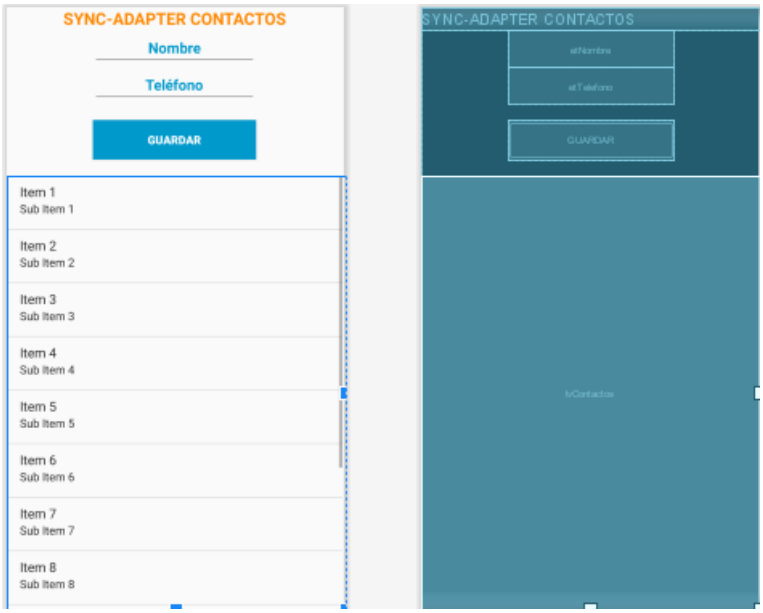

Ilustración 7. Vista previa del archivo activity_main.xml.

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<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:orientation="vertical"
    tools:context=".MainActivity">
    <!--Propiedades para el texto de encabezado-->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textStyle="bold"
        android:textColor="@android:color/holo_orange_dark"
        android:textSize="20sp"
        android:text="SYNC-ADAPTER CONTACTOS"/>
    <!--Propiedades para el campo de texto Nombre-->
    <EditText
        android:id="@+id/etNombre"
        android:layout_width="200dp"
        android:layout_gravity="center"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:gravity="center"
        android:textColorHint="@android:color/holo_blue_dark"
        android:textStyle="bold"
        android:hint="Nombre"/>
    <!--Propiedades para el campo de texto Teléfono-->
    <EditText
        android:id="@+id/etTelefono"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:inputType="phone"
        android:gravity="center"
        android:textColorHint="@android:color/holo_blue_dark"
        android:textStyle="bold"
        android:hint="Teléfono"/>
    <!--propiedades para el botón Guardar-->
    <Button
        android:id="@+id/btnGuardar"
        android:layout_width="200dp"
        android:layout_gravity="center"
        android:layout_margin="20dp"
        android:layout_height="wrap_content"
        android:background="@android:color/holo_blue_dark"
        android:textColor="@android:color/white"
        android:textStyle="bold"
        android:text="Guardar"/>
    <!--Propiedad para mostrar en un ListView-->
    <ListView
        android:id="@+id/lvContactos"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </ListView>

</LinearLayout>
```

Ilustración 8. Código archivo activity_main.xml.

Agregamos dependencias Para las solicitudes de red, usaremos Volley. Por lo tanto, agregamos la siguiente línea dentro del bloque de dependencias del archivo build.gradle de nivel de aplicación.

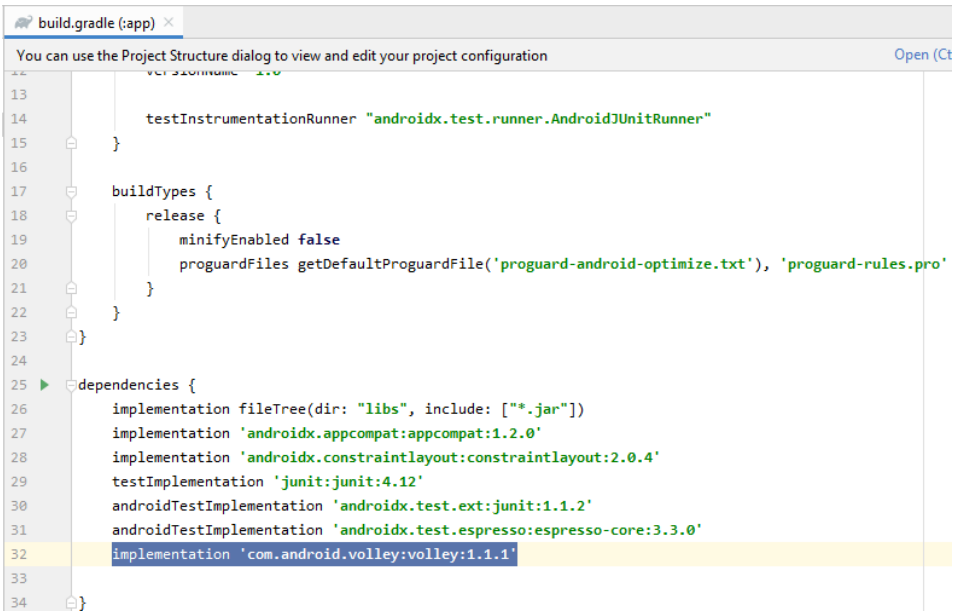

Ilustración 9. Agregar dependencias.

Creamos una nueva clase de tipo java donde vamos a crear la BD local de la clase SQLiteHopenHelper, creamos el nombre de la tabla y los nombres de las columnas; versión de la BD con su constructor al igual que el método para actualizar. Un método que toma dos argumentos, el primero es el nombre y teléfono que se guardará, el segundo es el estado 0 significa que el nombre y teléfono está sincronizado con el servidor 1 significa que el nombre y teléfono no está sincronizado con el servidor.

Un método que toma dos argumentos, el primero es el id del nombre y teléfono para el cual tenemos que actualizar el estado de sincronización y el segundo es el estado que se cambiará.

Un método que nos dará todo el nombre y teléfono almacenado en sqlite y otro método que es para obtener todos los nombres y teléfonos no sincronizados para que podamos sincronizarlo con la base de datos.

SQLiteDataHelper.java

```java
package com.example.syncadaptermysql;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class SQLiteDataHelper  extends SQLiteOpenHelper {
    //Constantes para el nombre de la base de datos, el nombre
    // de la tabla y los nombres de las columnas
    public static final String DB_NAME = "ContactosDB";
    public static final String TABLE_NAME = "usuario";
    public static final String COLUMN_ID = "id_usuario";
    public static final String COLUMN_NAME = "name";
    public static final String COLUMN_PHONE = "phone";
    public static final String COLUMN_STATUS = "status";
    //Versión de la BD
    private static final int DB_VERSION = 1;
    //Constructor de la BD
    public SQLiteDataHelper(Context context) { super(context, DB_NAME,  factory: null, DB_VERSION); }

    @Override
    //Creamos la BD
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE " + TABLE_NAME
                + "(" + COLUMN_ID +
                " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMN_NAME +
                " VARCHAR, " + COLUMN_PHONE +
                " VARCHAR, " + COLUMN_STATUS +
                " TINYINT);";
        db.execSQL(sql);
    }

    //Actualizar la base de datos
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        String sql = "DROP TABLE IF EXISTS Persons";
        db.execSQL(sql);
        onCreate(db);
    }
    //Este método toma dos argumentos, el primero es el nombre y teléfono que se guardará,
    // el segundo es el estado 0 significa que el nombre y teléfono está sincronizado con el
    // servidor 1 significa que el nombre y teléfono no está sincronizado con el servidor
    public boolean addName(String name, String telefono, int status) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();

        contentValues.put(COLUMN_NAME, name);
        contentValues.put(COLUMN_PHONE, telefono);
        contentValues.put(COLUMN_STATUS, status);

        db.insert(TABLE_NAME,  nullColumnHack: null, contentValues);
        db.close();
        return true;
    }
    //Este método toma dos argumentos, el primero es el id del nombre  y teléfono para el
```

Ilustración 10. Parte 1 del código SQLiteDataHelper.java.

```java
    // cual tenemos que actualizar el estado de sincronización y el segundo es
    // el estado que se cambiará
    public boolean updateNameStatus(int id, int status) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put(COLUMN_STATUS, status);
        db.update(TABLE_NAME, contentValues, whereClause: COLUMN_ID + "=" + id, whereArgs: null);
        db.close();
        return true;
    }
    //Este método nos dará todo el nombre y teléfono almacenado en sqlite
    public Cursor getNames() {
        SQLiteDatabase db = this.getReadableDatabase();
        String sql = "SELECT * FROM " + TABLE_NAME + " ORDER BY " + COLUMN_ID + " ASC;";
        Cursor c = db.rawQuery(sql, selectionArgs: null);
        return c;
    }
    //este método es para obtener todos los nombres y teléfonos no sincronizados
    //para que podamos sincronizarlo con la base de datos
    public Cursor getUnsyncedNames() {
        SQLiteDatabase db = this.getReadableDatabase();
        String sql = "SELECT * FROM " + TABLE_NAME + " WHERE " + COLUMN_STATUS + " = 0;";
        Cursor c = db.rawQuery(sql, selectionArgs: null);
        return c;
    }
}
```

Ilustración 11. Parte 2 del código SQLiteDataHelper.java.

Vamos a utilizar Volley para la solicitud http. Entonces para esto crearemos una clase singleton. Creamos una clase llamada VolleySingleton y escribimos el siguiente código.

VolleySingleton.java

```java
package com.example.syncadaptermysql;

import android.content.Context;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.toolbox.Volley;


public class VolleySingleton {
    private static VolleySingleton mInstance;
    private RequestQueue mRequestQueue;
    private static Context mCtx;

    private VolleySingleton(Context context) {
        mCtx = context;
        mRequestQueue = getRequestQueue();

    }

    public static synchronized VolleySingleton getInstance(Context context) {
        if (mInstance == null) {
            mInstance = new VolleySingleton(context);
        }
        return mInstance;
    }
```

Ilustración 12. Parte 1 del código VolleySingleton.java.

```
27
28      public RequestQueue getRequestQueue() {
29          if (mRequestQueue == null) {
30              // getApplicationContext () es clave, evita que se filtre el
31              // Activity o BroadcastReceiver si alguien pasa uno.
32              mRequestQueue = Volley.newRequestQueue(mCtx.getApplicationContext());
33          }
34          return mRequestQueue;
35      }
36
37      public <T> void addToRequestQueue(Request<T> req) {
38          getRequestQueue().add(req);
39      }
```

Ilustración 13. Parte 2 del código VolleySingleton.java.

Ahora crearemos una clase modelo Name.java con las variables a llamar con sus getters.

Name.java

```
1   package com.example.syncadaptermysql;
2
3   public class Name {
4       //Variables que mandamos a llamar
5       private String name;
6       private int status;
7       private String phone;
8
9       public Name(String name, String phone, int status) {
10          this.name = name;
11          this.status = status;
12          this.phone = phone;
13      }
14      //Creación de getters
15      public String getName() {
16          return name;
17      }
18
19      public int getStatus() {
20          return status;
21      }
22
23      public String getPhone(){
24          return phone;
25      }
26  }
```

Ilustración 14. Clase Modelo.

Ahora creamos una clase de tipo java NetworkStateChecker.java que detectara el estado de la red y creamos un receptor de transmisión.

NetworkStateChecker.java

```java
package com.example.syncadaptermysql;

import ...

public class NetworkStateChecker extends BroadcastReceiver {
    private Context context;
    private SQLiteDataHelper db;

    @Override
    public  void onReceive(Context context, Intent intent){
        this.context = context;
        db = new SQLiteDataHelper(context);

        ConnectivityManager cm = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

        //Si existe la conexión...
        if (activeNetwork != null) {
            //Si está conectado (wi-fi o datos)
            if (activeNetwork.getType() == ConnectivityManager.TYPE_WIFI || activeNetwork.getType() == ConnectivityManager.TYPE_MOBILE) {

                //Obtiene todos los datos no sincronizados
                Cursor cursor = db.getUnsyncedNames();
                if (cursor.moveToFirst()) {
                    do {
                        //Guarda los datos no sincronizados
                        saveName(
                                cursor.getInt(cursor.getColumnIndex(SQLiteDataHelper.COLUMN_ID)),
                                cursor.getString(cursor.getColumnIndex(SQLiteDataHelper.COLUMN_NAME)),
                                cursor.getString(cursor.getColumnIndex(SQLiteDataHelper.COLUMN_PHONE))
                        );
                    } while (cursor.moveToNext());
                }
            }
        }
    }

    private void saveName(final int id, final String name, final String phone) {
        StringRequest stringRequest = new StringRequest(Request.Method.POST, MainActivity.URL_SAVE_DATA,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        try {
                            JSONObject obj = new JSONObject(response);
                            if (!obj.getBoolean( name: "error")) {
                                //Actualizar el estado en sqlite
                                db.updateNameStatus(id, MainActivity.NAME_SYNCED_WITH_SERVER);

                                //Enviando la transmisión para actualizar la lista
                                context.sendBroadcast(new Intent(MainActivity.DATA_SAVED_BROADCAST));
                            }
                        } catch (JSONException e) {
                            e.printStackTrace();
                        }
                    }
                },
                    new Response.ErrorListener() {
                        @Override
                        public void onErrorResponse(VolleyError error) {


                        }
                    }) {
                @Override
                //Mapear los valores que se van a mandar y almacenar los parametros
                protected Map<String, String> getParams() throws AuthFailureError {
                    Map<String, String> params = new HashMap<>();
                    params.put( k: "name", name);
                    params.put( k: "phone", phone);
                    return params;
                }
            };//Ejecutamos la clase para la libreria Volley
            VolleySingleton.getInstance(context).addToRequestQueue(stringRequest);
        }
    }
```

Ilustración 15. Código del archivo NetworkStateChecker.java.

Creamos un adaptador para nuestra LisView NameAdapter.java.

NameAdapter.java

```java
package com.example.syncadaptermysql;

import ...

public class NameAdapter extends ArrayAdapter<Name> {

    //Para almacenar todos los nombres
    private List<Name> names;
    private Context context;

    //Constructor de la clase
    public NameAdapter(Context context, int resource, List<Name> names){
        super(context, resource, names);
        this.context = context;
        this.names = names;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent){
        LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        //Para obtener los items
        View listViewItem = inflater.inflate(R.layout.names,  root: null,  attachToRoot: true);
        TextView textViewName = (TextView) listViewItem.findViewById(R.id.textViewName);
        ImageView imageViewStatus = (ImageView) listViewItem.findViewById(R.id.imageViewStatus);

        //Obteniendo el nombre y teléfono actual
        Name name = names.get(position);

        //Colocando el nombre y teléfono en el textView
        textViewName.setText(name.getName()+ " - "+name.getPhone());

        //Para asignar el icono del registro de acuerdo a si está sincronizado o no
        if (name.getStatus() == 0 )
            imageViewStatus.setBackgroundResource(R.drawable.stopwatch);
        else
            imageViewStatus.setBackgroundResource(R.drawable.success);
        return listViewItem;
    }
}
```

Ilustración 16. Código del archivo NameAdapter.java.

Ahora en el MainActivity.java colocamos el siguiente Código:

Creamos una variable para la URL con la dirección IP y archivo alojado en XAMPP, Inicialización de la instancia de la BD, oyente del botón guardar, llamada al método para cargar todos los nombres almacenados, el receptor de transmisión para actualizar el estado de sincronización, registrando el broadcast receiver para actualizar el status de la sincronización.

MainActivity.java

```java
package com.example.syncadaptermysql;

import ...

public class MainActivity extends AppCompatActivity {
    //Variable para guardar los valores con la direccion IP y archivo alojado
    //en el servidor de XAMPP
    public static final String URL_SAVE_DATA = "http://192.168.52.1/WebService/save.php";
    SQLiteDataHelper db;//objeto para la bd
    //Creación de valiables para los objetos
    Button btnGuardar;
    EditText etNombre;
    EditText etTelefono;
    ListView lvContactos;
    //Lista para almacenar todos los nombres
    private List<Name> names;


    //1 significa que los datos estan sincronizados y 0 que no
    public static final int NAME_SYNCED_WITH_SERVER = 1;
    public static final int NAME_NOT_SYNCED_WITH_SERVER = 0;

    //Un receptor para saber si los datos estan sincronizados o no
    public static final String DATA_SAVED_BROADCAST = "com.ricardo.datasaved";

    //Broadcast receiver para saber si el status de la sincronización
    private BroadcastReceiver broadcastReceiver;

    //Adapterobject para el ListView
    private NameAdapter nameAdapter;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        registerReceiver(new NetworkStateChecker(), new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));
        //Inicialización de la instancia de la BD
        db = new SQLiteDataHelper( context: this);
        names = new ArrayList<>();
        //Linkeo de controladores
        btnGuardar = (Button) findViewById(R.id.btnGuardar);
        etNombre = (EditText) findViewById(R.id.etNombre);
        etTelefono = (EditText) findViewById(R.id.etTelefono);
        lvContactos= (ListView) findViewById(R.id.lvContactos);

        //Oyente del botón Guardar
        btnGuardar.setOnClickListener((v) -> { saveNameToServer(); });

        //Lamada al método para cargar todos los nombres almacenados
        loadNames();

        // El receptor de transmisión para actualizar el estado de sincronización
        broadcastReceiver = (BroadcastReceiver) (context, intent) -> {

                //Cargando los nombres y teléfonos de nuevo
                loadNames();
        };

        //Registrando el broadcast receiver para actualizar el status de la sincronización
        registerReceiver(broadcastReceiver, new IntentFilter(DATA_SAVED_BROADCAST));
    }
    //este método cargará los nombres de la base de datos con
    // el estado de sincronización actualizado
    private void loadNames() {
        names.clear();
        Cursor cursor = db.getNames();
        if (cursor.moveToFirst()) {
```

Ilustración 17. Parte 1 del código MainActivity.java.

Actualizamos la BD y guardar el nombre y teléfono en el servidor; así como hacer una validación de campos vacíos al presionar el botón guardar. Si hay éxito almacenara el nombre y teléfono con estado sincronizado; si no guarda exitosamente se almacena en la BD SQLite con el estado no sincronizado.

```
107             do {
108                 Name name = new Name(
109                         cursor.getString(cursor.getColumnIndex(SQLiteDataHelper.COLUMN_NAME)),
110                         cursor.getString(cursor.getColumnIndex(SQLiteDataHelper.COLUMN_PHONE)),
111                         cursor.getInt(cursor.getColumnIndex(SQLiteDataHelper.COLUMN_STATUS))
112                 );
113                 names.add(name);
114             } while (cursor.moveToNext());
115         }
116 
117         nameAdapter = new NameAdapter( context: this, R.layout.names, names);
118         lvContactos.setAdapter(nameAdapter);
119     }
120 
121     //Este método simplemente actualizará la lista
122     private void refreshList() { nameAdapter.notifyDataSetChanged(); }
125     //Este método es guardar el nombre en el servidor
126     private void saveNameToServer() {
127         final ProgressDialog progressDialog = new ProgressDialog( context: this);
128 
129         final String name = etNombre.getText().toString().trim();
130         final String phone = etTelefono.getText().toString().trim();
131 
132         if (phone.isEmpty() || name.isEmpty()){
133             Toast.makeText(getApplicationContext(),  text: "No puede dejar campos vacios", Toast.LENGTH_SHORT).show();
134         }
135         else {
136             progressDialog.setMessage("Saving Name...");
137             progressDialog.show();
138             StringRequest stringRequest = new StringRequest(Request.Method.POST, URL_SAVE_DATA,
139                     new Response.Listener<String>() {
140                         @Override
141                         public void onResponse(String response) {
142                             progressDialog.dismiss();
143                             try {
144                                 JSONObject obj = new JSONObject(response);
145                                 if (!obj.getBoolean( name: "error")) {
146                                     // si hay un exito
147                                     // almacenando el nombre en sqlite con estado sincronizado
148                                     saveNameToLocalStorage(name, phone, NAME_SYNCED_WITH_SERVER);
149                                 } else {
150                                     //Si no guarda exitosamente se almacena en la BD
151                                     // SQLite con el status no sincronizado
152                                     saveNameToLocalStorage(name, phone, NAME_NOT_SYNCED_WITH_SERVER);
153                                 }
154                             } catch (JSONException e) {
155                                 e.printStackTrace();
156                             }
157                         }
158                     },
159                     new Response.ErrorListener() {
160                         @Override
161                         public void onErrorResponse(VolleyError error) {
162                             progressDialog.dismiss();
163                             //En caso de error al almacenar el nombre en sqlite con estado no sincronizado
164                             saveNameToLocalStorage(name, phone, NAME_NOT_SYNCED_WITH_SERVER);
```

Ilustración 18. Parte 2 del código MainActivity.java.

Mapea los valores que se van a mandar y almacenar.

```
                    }
                }) {
            @Override
            protected Map<String, String> getParams() throws AuthFailureError {
                Map<String, String> params = new HashMap<>();
                params.put( k: "name", name);
                params.put( k: "phone", phone);
                return params;
            }
        };
        //Ejecutamos la clase para la libreria Volley
        VolleySingleton.getInstance(this).addToRequestQueue(stringRequest);
    }
}//Mapear los valores que se van a mandar y almacenar los parametros
private void saveNameToLocalStorage(String name, String phone, int status) {
    etNombre.setText("");
    etTelefono.setText("");
    db.addName(name, phone, status);
    Name n = new Name(name, phone, status);
    names.add(n);
    refreshList();
}
}
```

Ilustración 19. Parte 3 del código MainActivity.java.

Creamos un layout de tipo xml llamado name.xml donde se imprimirán los datos en un TextView y verifica con una imagen si estan guardados los datos o no estan sincronizados.

SQLiteDataHelper.java | names.xml | MainActivity.java

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:text="Name   "
        android:layout_alignParentLeft="true"
        android:id="@+id/textViewName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <ImageView
        android:id="@+id/imageViewStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:background="@drawable/success" />
</RelativeLayout>
```

Ilustración 20. Código del archivo names.xml.

Resultados

Ejecutamos nuestra aplicación en el emulador de nuestra maquina y vemos la pantalla principal. En la otra imagen vemos los datos que están de prueba en la BD en nuestro servidor XAMPP.

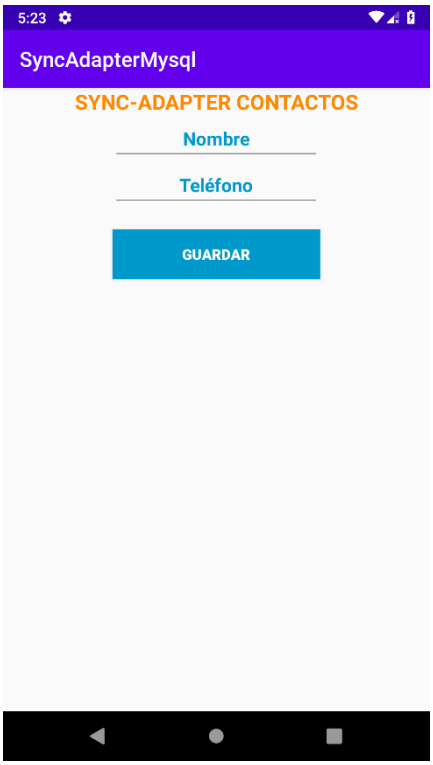

Ilustración 21. Parte 1 del resultado de la aplicación móvil.

Ahora agregamos un registro para verificar que se inserten, pero antes presionamos el botón guardar sin insertar datos y nos manda un mensaje de validación de campos vacíos. Ahora agregamos un registro y nos manda un mensaje de guardando registro y por último vemos que ya se guardó el registro.

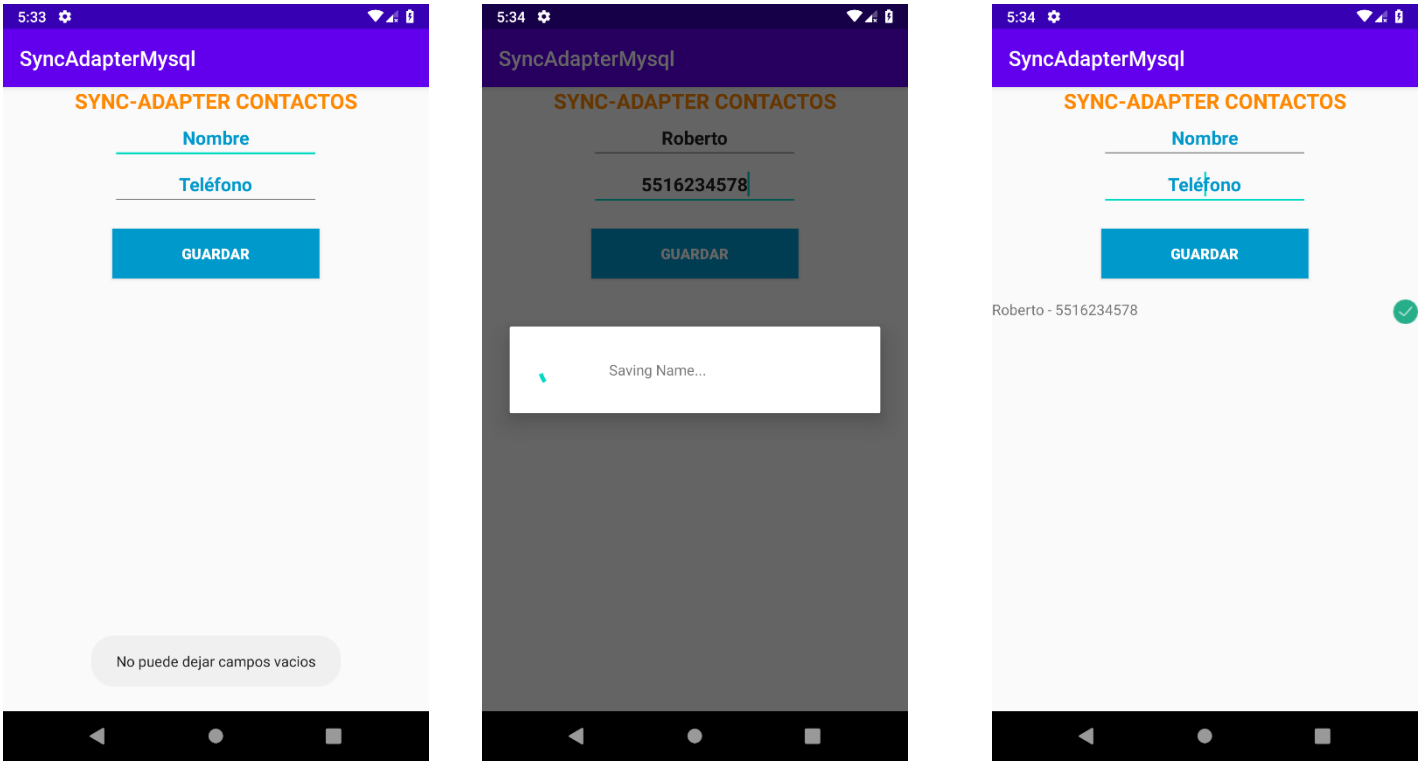

Ilustración 22, Parte 2 del resultado de la aplicación móvil.

Vamos a nuestra BD y refrescamos y ya debe aparecer nuestro registro nuevo.

Ilustración 23. Registro insertado.

Ahora hacemos un nuevo registro, pero apagamos nuestra red de Wi-Fi para ver el resultado; nos mandara una imagen con el estado de no registrado podemos volver encender el Wi-Fi y el estado de la imagen cambia ha guardado.

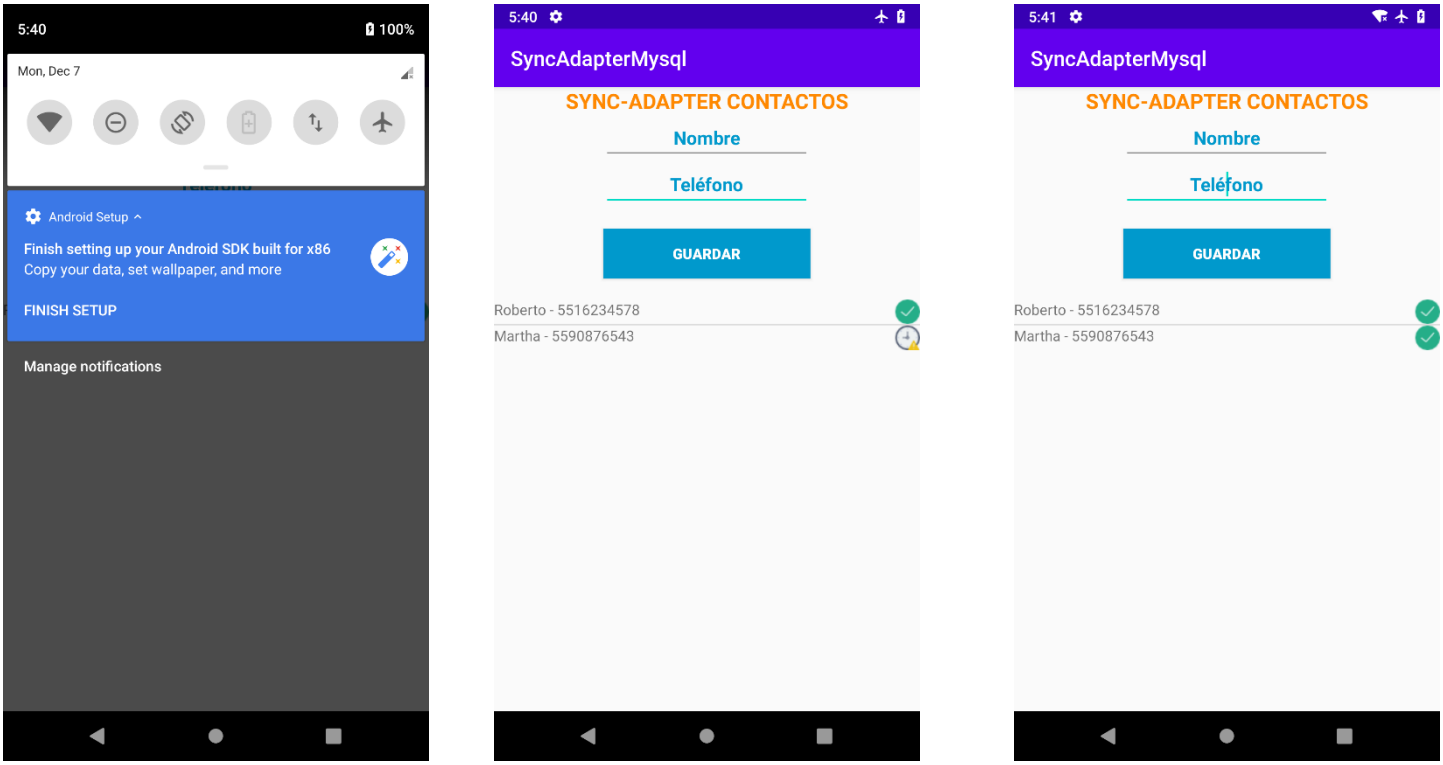

Ilustración 24. Parte 3 del resultado de la aplicación móvil.

Ahora vemos nuestro nuevo registro insertado en la BD.

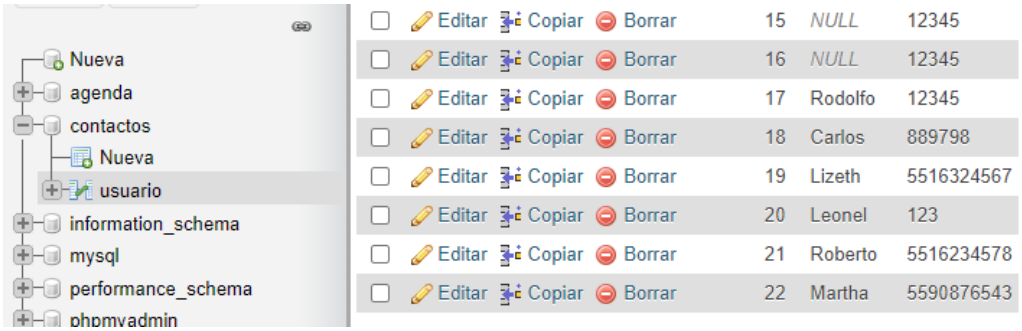

Ilustración 25. Nuevo registro insertado.

Conclusión

En esta práctica vimos cómo crear una practica de Sync-Adapter MSQLite en MySQL con la librería VOLLEY donde al principio se tenían dudas, se buscó información acerca de la practica y obtuvimos los resultados que se deseaban; teniendo como finalidad el insertar un registro y si no hay conexión con nuestro adaptador de red no los guarda en nuestra base de datos alojada en nuestro servidor local (XAMPP), se guardan en tiempo de espera en la base de datos del móvil; una vez encendido el adaptador de red instantáneamente los guarda en nuestra BD por medio del WebServices que manda los datos en formato Json.