2020

IES
LEONARDO
DA VINCI

Estudiante:
Vasile Claudiu
Ungur

[PROYECTO GRÁFICAS]

FP DE GRADO SUPERIOR – DESARROLLO DE APLICACIONES MULTIPLATAFORMA

INDICE

1. Descripción del proyecto

El proyecto consiste en una aplicación Android que es capaz de conectarse a un servidor remoto (en internet o en el ordenador de casa) donde reside una base de datos con los parámetros recibidos de unos sensores instalados en un inmueble, bien sea una casa, un edificio o incluso un invernadero.

Dichos sensores miden factores como la temperatura, humedad, Nivel CO2, o detectan que luces están encendidas o que ventanas o puertas están abiertas etc. (Los sensores se han simulado, debido a los costes que supondría instalarlos físicamente).

La aplicación presentará en formato de graficas los valores existentes en la base de datos, a partir de la fecha y hora indicadas por el usuario, y lo hará teniendo en cuenta tanto los últimos valores o los actuales, como los anteriores, es decir que el usuario dispondrá de un histórico de dichos datos, lo que supone una serie de ventajas si se quiere trabajar con dicha información (hacer análisis o comparativas por ejemplo).

También es posible elegir que parámetros quieren ser visualizados, así como también disponer de una alerta configurable para los sensores de movimiento, luz exterior, luz del salón, ventana y puertas, así por ejemplo podemos saber si una puerta ha sido abierta, o hay movimiento en una de las estancias del inmueble por ejemplo.

2. Descripción del entorno de trabajo: Condiciones de la empresa a la que se adapta el proyecto.

La aplicación diseñada es flexible puesto que puede adaptarse a distintos tipos de empresa, desde una casa particular, una oficina, una empresa con maquinaria industrial o hasta un invernadero. Esto es posible porque la aplicación cuenta con sensores de movimiento, temperatura, humedad, nivel de CO2, luces, apertura de ventanas y puertas.

Sensores que como hemos dicho, pueden adaptarse a diferentes cometidos, desde los más sencillos como puede ser una casa, donde nos ofrece seguridad, en el sentido de que nos permite saber si se ha producido movimiento o hay alguien en una zona donde en ese momento no debería haber nadie, saber si se han abierto puertas o ventanas o controlar el nivel de CO2.

En una empresa con maquinaria industrial, además de cuestiones de seguridad como en el caso de la vivienda, nos ofrece también la posibilidad de monitorizar maquinaria industrial, así por ejemplo saber cuándo una máquina esta en uso o no y si debería estar

en uso actuar en consecuencia, o si se trata de una empresa con invernaderos por ejemplo podemos controlar los niveles tanto de humedad como de gases o controlar que las puertas estén cerradas.

Otra cuestión muy importante a resaltar, es que contamos con una base de datos históricos que nos permite tener almacenada toda esa información, es decir nos permite acceder a ella de forma rápida, sencilla y desde cualquier lugar, y si lo que queremos es trabajar con esa información, las posibilidades aumentan y pueden ser muy diversas, así por ejemplo podemos usarla para formular análisis o mejorar los procesos productivos de la empresa, por ejemplo en el caso del invernadero cual es el punto de humedad óptimo para obtener mejores resultados o en el caso de la empresa con maquinaria industrial, comprobar en qué tiempo las máquinas están a pleno rendimiento, cuales son las franjas horarias donde los operarios presentan un mayor nivel de productividad, teniendo así más información que pueda ayudar a mejorar los procesos productivos.

Y por último en el ámbito de la seguridad, mencionar también que podemos configurar una alerta en los sensores de movimiento, luces, y apertura de ventanas y puertas, en el caso de que quisiéramos conocer exactamente si se produce un cambio y nosotros no estamos en el edificio en ese momento. Además nos ofrece también toda esta información de una forma visual en el sentido de que podemos extraerla en tres tipos de gráficas (LineChart, BarChart y BarLineChart).

3. Estudio de necesidades

Las empresas en su día a día, tienen la necesidad gestionar su organización y para hacerlo de forma eficiente, es clave la toma de decisiones, y para lograr tomar las decisiones adecuadas un factor clave es aprovechar la información y disponer del conocimiento de su propio contexto, de este modo la organización puede tener la capacidad de elegir la acción más favorable para su desarrollo.

La información y conocimiento de su propio contexto en una empresa se convierte en un recurso muy importante a la hora de llevar a cabo sus funciones diarias para lograr un alto nivel competitivo, además de procurar su crecimiento. Tener la información a su alcance para ser analizada se convierte en una necesidad y en este punto las herramientas tecnológicas juegan un papel fundamental para integrar todos esos datos y aumentar el valor que ellos mismos aportan a la empresa.

A pesar de que la información se genera de forma automática durante el desarrollo de los procesos empresariales muchas empresas no están aprovechando esa información

para una correcta toma de decisiones, en este sentido la aplicación diseñada puede ser de mucha utilidad si la empresa empieza decide aprovechar este recurso.

4. Recursos necesarios:

Los medios necesarios para la realización del proyecto son:

Hardware:

- ✓ Un ordenador con conexión a internet para la instalación de los servidores y de la base de datos
- ✓ Un dispositivo móvil con Android
- ✓ Sensores

Software

- ✓ La plataforma Android Studio
- ✓ Un servidor web y un servidor MySql instalados en el PC
- ✓ Sistema para leer los sensores y escribirlos en la base de datos (Rasberry pi por ejemplo.

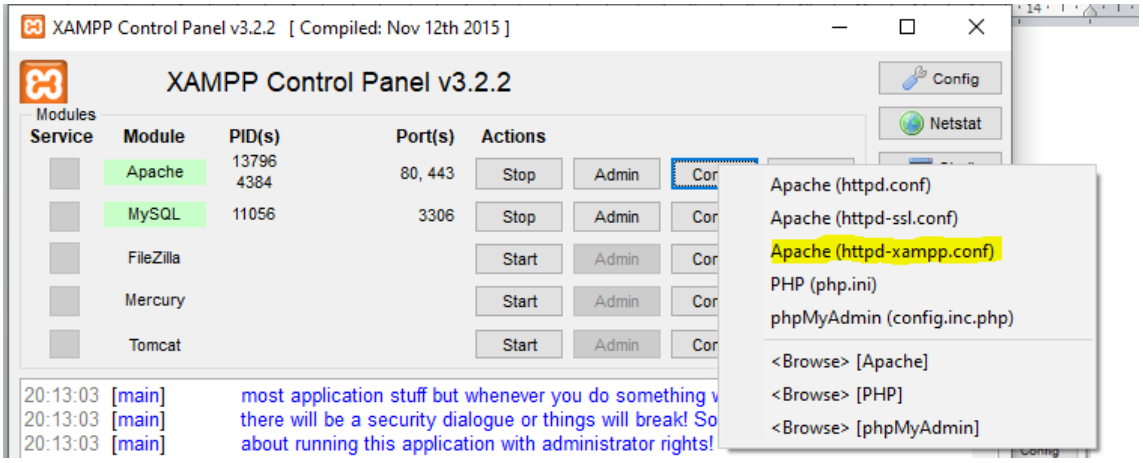
Conexión wifi: Ya que el enrutamiento desde el teléfono móvil se realiza (en este caso) a través de la LAN del rúter para poder alcanzar el servidor Web, y de esta forma no es necesario reconfigurar el rúter y el servidor web para su alcance a través de internet. No obstante la aplicación funcionaria si el servidor web estuviese publicado en los DNS de internet, es decir que funcionaria en un entorno real.

5. Propuesta técnica

5.1 Instalación de xamp

En primer lugar, descargamos xamp desde una página segura de internet, xamp es un paquete de software libre que básicamente consiste en un sistema de gestión de base de datos MySQL, el servidor Apache y los intérpretes para el lenguaje PHP

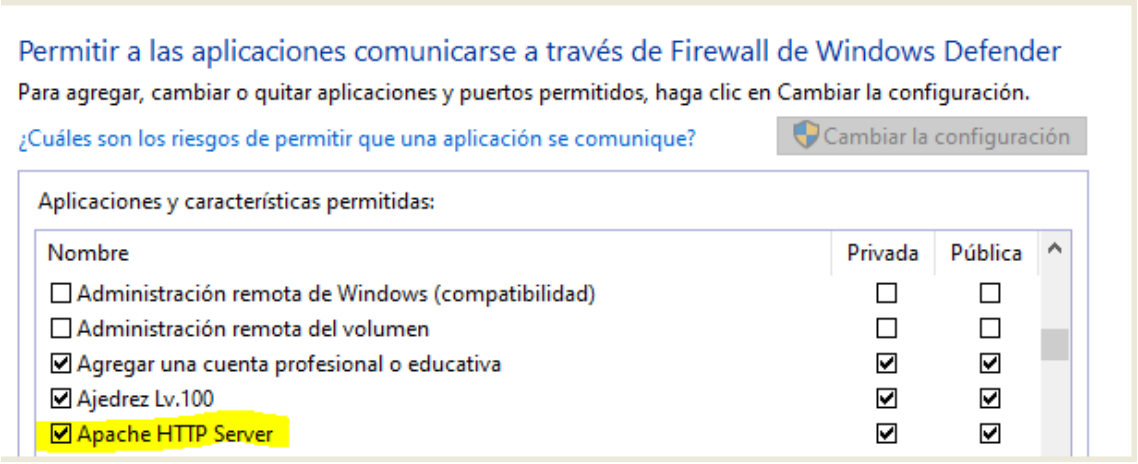
Una vez instalado, desde el panel de control de xamp, del botón Config de Apache abrimos el fichero de configuración apache (httpd-xamp.conf).



Reconfiguramos la sección de Seguridad para poder alcanzar el servidor tanto desde localhost como desde cualquier IP, sea desde La LAN o desde Internet si el servidor Web se publica en los DNS de internet:

5.2 Configuración del firewall del ordenador de usuario

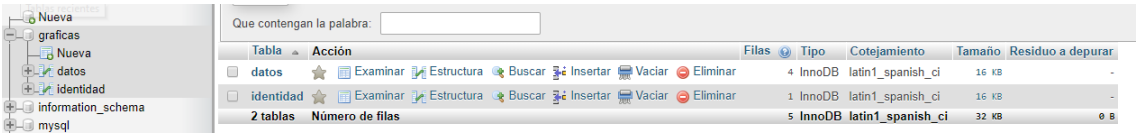
Ahora se debe configurar el firewall del PC para permitir la comunicación del servidor Apache:



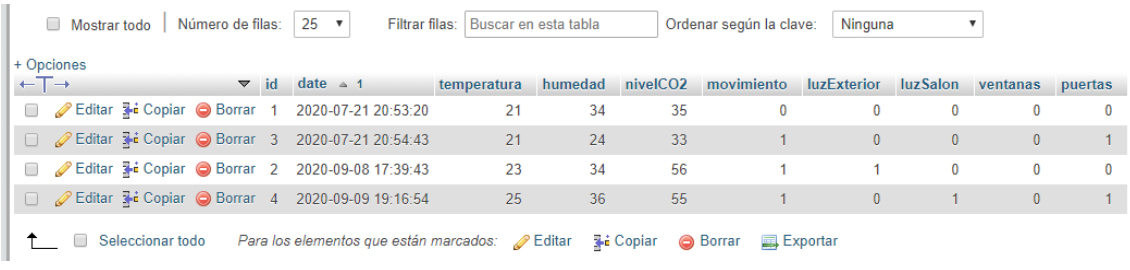
5.3 Creación de la base de datos

Ahora ya se puede acceder por navegador al servidor MySql y se crea aquí la base de datos:

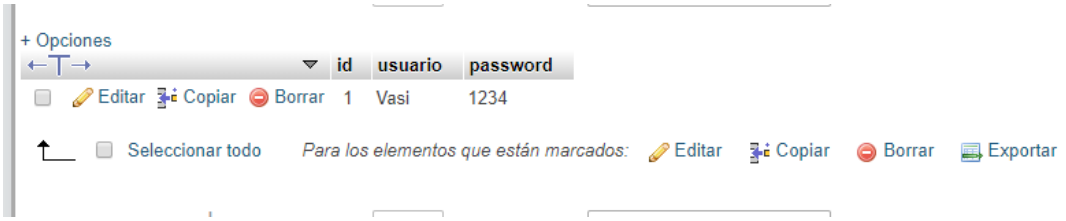
Creamos las tablas necesarias para el proyecto poniendo unos campos de prueba



En la tabla datos vamos a almacenar toda la información que llegará desde los sensores:



En la tabla identidad, se almacenarán las credenciales del usuario



5.4 Configuración de los ficheros php

Hecho lo anterior, ya se pueden configurar los ficheros php necesarios para la comunicación con la aplicación Android. Los ficheros se colocan en una carpeta del directorio htdocs de xampp: C:\xampp\htdocs\ProyectoGraficas

Los ficheros necesarios son:

- ✓ **login.php** : Que recibe el desde la Activity Login de la aplicación Android el usuario y password para la conexión al servidor MySql, este fichero comprueba esta información en la tabla identidad del Sql y devuelve un campo success en true o false en función de la respuesta del servidor. La información se envía a la aplicación en formato json:

```php
<?php
$hostname='localhost';
$database='graficas';
$username='root';
$password='';
$con = new mysqli($hostname,$username,$password,$database);

$user  = $_POST["usuario"];
$pass  = $_POST["clave"];
$statement =mysqli_prepare($con,"SELECT * FROM identidad WHERE usuario = ? AND password = ?");
mysqli_stmt_bind_param($statement,"ss",$user,$pass);
mysqli_stmt_execute($statement);

mysqli_stmt_store_result($statement);
mysqli_stmt_bind_result($statement,$idusuario,$usuario,$clave);
 //La respuesta de la base de datos la codifico en json
 //El parametro que se envia se llama "success" y es de tipo boolean
$response =array();
$response["success"] = false;

while(mysqli_stmt_fetch($statement)){
  $response["success"]=true;
  $response["usuario"]=$usuario;
  $response["clave"]=$clave;
}
  //Se responde en formato json
echo json_encode($response);
?>
```

- ✓ **configuracion.php:** Que recibe el desde la Activity Configuracion de la aplicación Android una variable fechayhora, y se conecta al servidor MySql, para recoger los datos de la tabla datos a partir de la fecha y hora indicada.

Devuelve esta información en un array, junto con un campo success en true o false en función de la respuesta del servidor, que servirá para el dialogo entre usuario y la aplicación. La información se envía a la aplicación en formato json:

```php
$ar["success"]=false;
// Se inicializa el array as a null por si no hay datos para que no presente un warning
$as=null;
while ($fila = $consulta->fetch_assoc()) {
     $i++;
     //Se pone a true por recojer el valor booleano en java
     $ar["success"]=true;
     //Se añaden los campos a cada array
     $date["fecha".$i]=$fila['date'];
     $temp["temperatura".$i]=$fila['temperatura'];
     $hum["humedad".$i]=$fila['humedad'];
     $codos["nivelCO2".$i]=$fila['nivelCO2'];
     $mov["movimiento".$i]=$fila['movimiento'];
     $luzext["luzExterior".$i]=$fila['luzExterior'];
     $luzsalon["luzSalon".$i]=$fila['luzSalon'];
     $vent["ventanas".$i]=$fila['ventanas'];
     $puert["puertas".$i]=$fila['puertas'];
     $ar["num"]=$i;
     //Se concatenan los arrays en el array as
     //Por cada bucle se cambia el valor del $ar
     $as=array_merge($ar,$date,$temp,$hum,$codos,$mov,$luzext,$luzsalon,$vent,$puert);

}
 if($as==null){
 $as=$ar;
 }
//Se responde en formato json

 echo json_encode($as);
```

- ✓ **consulta.php:** Que se conecta al servidor MySql y a través de consulta sql, recibe los datos de la última fecha de la tabla datos y los guarda en un array que se devolverá a la clase MyReceiver de la aplicación, estos datos son necesarios para la alarma de la aplicación.

```php
$database='graficas';
$username='root';
$password='';
$con = new mysqli($hostname,$username,$password,$database);

if ($con->connect_errno) {
    echo "Falló la conexión a MySQL: (" . $con->connect_errno . ") " . $con->connect_error;
}

$consulta = $con->query("SELECT * FROM `datos` WHERE `date` =  (SELECT MAX(`date`) from `datos`)");
$i=0;
 // Se inicializa el array as a null por si no hay datos para que no presente un warning
$as=null;
while ($fila = $consulta->fetch_assoc()) {
      $i++;
     //Se añaden los campos a cada array

     $mov["movimiento"]=$fila['movimiento'];
     $luzext["luzExterior"]=$fila['luzExterior'];
     $luzsalon["luzSalon"]=$fila['luzSalon'];
     $vent["ventanas"]=$fila['ventanas'];
     $puert["puertas"]=$fila['puertas'];
     $ar["num"]=$i;
     //Se concatenan los arrays en el array as
     $as=array_merge($ar,$mov,$luzext,$luzsalon,$vent,$puert);

}
 if($as==null){
 $as=$ar;
 }
//Se responde en formato json
```

5.4.1 Ficheros Adicionales:

Adicionalmente he creado otros 2 ficheros php: formulariosDatosBBDD.php y insertarDatosBBDD.php que ayudan a insertar nuevas filas en la base de datos, simulando de esta forma los datos que recibe la tabla desde los sensores.

5.5 Creación de la aplicación en la plataforma Android:

El siguiente paso es crear la aplicación en la plataforma Android Studio. He creado las siguientes clases:

- MainActivity, Login,Configuracion, Chart y BarLineChart extendidas a AppCompatActivity,
- DatePickerDialog y TimePickerFragment extendidas a DialogFragment para la selección de la fecha y hora,
- MyReceiver extendida a android.content.BroadcastReceiver
- AuxAlarma extendida a IntentService,
- Auxiliar extendida a StringRequest y MyAlertDialog extendida a Activity

Los layout necesarios son: activity_main.xml, activity_login.xml, activity_configuracion.xml, activity_chart.xml, activity_bar_line_chat.xml y activity_my_alert_dialog.xml el cual será transparente.

Otros ficheros son: menú.xml en el directorio menú, alert.mp3 en el directorio raw, array_anime.xml en values, y unas fotos utilizadas para la presentación.

5.6. Descripción de las clases y el funcionamiento:

Para empezar se debe proporcionar el permiso necesario para la conexión a internet en el fichero manifest:

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

En el fichero build.gradle de la app se añade el repositorio:

```
repositories {

        maven { url "https://jitpack.io" }
}
```

Se implementa:

`implementation 'com.android.volley:volley:1.1.0'` para la comunicación con los ficheros php del Apache y

`implementation 'com.github.PhilJay:MPAndroidChart:v3.0.3'` para las gráficas.

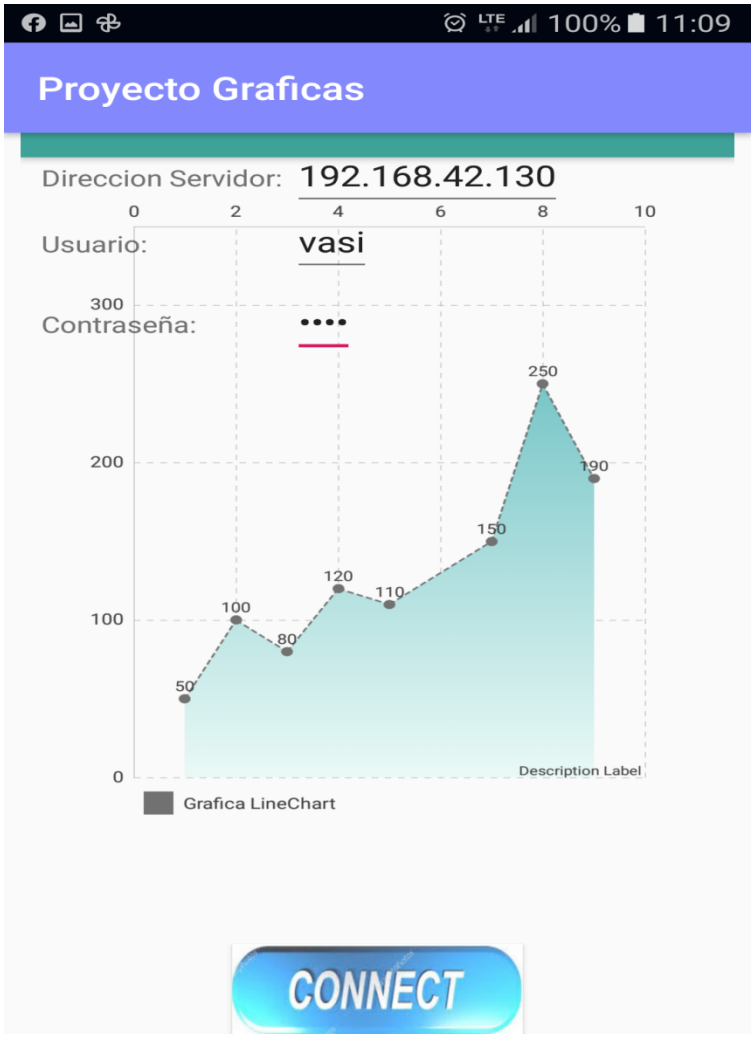
Las demás implementaciones vienen por defecto al crear el proyecto.

5.6.1 Clase MainActivity:

En esta clase se describe el funcionamiento del programa, y a través de un Intent, al pulsar el botón Empezar, se abre la Activity Login

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

    Button btn = (Button) findViewById(R.id.empesar);
    btn.setOnClickListener((v) -> {
            Intent intent = new Intent (v.getContext(), Login.class);
            startActivityForResult(intent,  requestCode: 0);
    });
}
```

5.6.2 Clase Login:



En esta clase, a través de 3 EditText de la interfaz, se recogen la dirección del servidor, usuario y la contraseña y se le pasa al constructor correspondiente de la clase Auxiliar para la comunicación con el servidor Apache.

El paso al constructor se realiza en la llamada al constructor:

```
//Llamo al constructor de la clase Auxiliar
Auxiliar r = new Auxiliar(usu,pass,ruta,respuesta,error);
```

Se puede observar que se envía los Strings usu,pass, que son el usuario y el password introducidos por el teclado, recogidos en:

```
//Se recupera la informacion de la interfaz
String usu = usuarioLog.getText().toString();
String pass = claveLog.getText().toString();
```

Tambien se le pasa la ruta al fichero login.php. La Ruta es :

```
server = dirServer.getText().toString();
String ruta = "http://"+server+"/ProyectoGraficas/login.php";//Se
configura la ruta al servidor
```

Aquí, además de recoger en String la dirección introducida por el teclado, se le añade la ruta exacta al fichero deseado.

Desde el constructor se espera la respuesta desde el servidor que se define en el código:

```
//Respuesta por parte del RegistroRequest al pasar los datos
 Response.Listener<String> respuesta = new Response.Listener<String>()
{
```

También se espera un código de error que se trata en el siguiente código, y se presenta la Excepción, según el error encontrado:

```
//Respuestas según el error de conexión
 Response.ErrorListener error= new Response.ErrorListener() {
     @Override
     public void onErrorResponse(VolleyError error) {
         String message ="";
         if (error instanceof NetworkError) {
             message = "El servidor indicado no puede ser encontrado";
         } else if (error instanceof ServerError) {
             message = "Actualmente hay un problema con el servidor,
```

```
Por favor intente mas tarde";
        } else if (error instanceof NoConnectionError) {
            message = "Hay un problema con la conexion a internet.
Por favor revise tu conexion a internet";
        } else if (error instanceof TimeoutError) {
            message = "El tiempo de espera ha sido superado.El
servidor no responde.";
        }
        displayExceptionMessage("ERROR DE CONEXION : "+message);
    }
 };
```

La respuesta se guarda en una cola de peticiones Volley.*newRequestQueue* que serán tratadas en orden de llegadas, en el bloque try/catch del método onResponse

```
// Utilizo Volley para la comunicacion
RequestQueue cola = Volley.newRequestQueue(Login.this);
cola.add(r);
```

Cada respuesta (si fuesen más de una) entrara en el método onResponse donde se crea un JSONObject para la respuesta del servidor, y según la variable success (boolean) recibida desde el servidor, se presenta un dialogo para reintentar la conexión en el caso de que falle la autenticación. El dialogo se crea a través de AlertDialog.Builder, que presenta un mensaje de fallo y un botón para reintentar. Una vez pulsado el botón, devuelve esta Activity al inicio, manteniendo los campos introducidos por el usuario.

```
if (ok==true){

.........

}else{
    conectado=true;
    //Se crea la alerta
    AlertDialog.Builder alerta = new AlertDialog.Builder(Login.this);
    alerta.setMessage("Fallo del usuaro o contraseña ")
            .setNegativeButton("Reintentar", null)
            .create()
            .show();
}
```

En el caso de que la conexión sea exitosa, incluyendo usuario y password, se abre la clase Configuración a través de un Intent, pasándole la dirección del servidor, y después se cierra esta clase. De esta forma el usuario no necesita volver a introducir la dirección del servidor en las siguientes Activitys

```
if(ok==true){

    displayExceptionMessage("CONEXIÓN EXITOSA");

    //Atraves del intent pasa la informacion a la clase Configuracion
    Intent conf = new Intent (Login.this, Configuracion.class);
    // Se pasa a la clase Configuracion el servidor introducido
    conf.putExtra("server",server);
    //Aranca la actividad Configuracion
    Login.this.startActivity(conf);
    Login.this.finish();
```

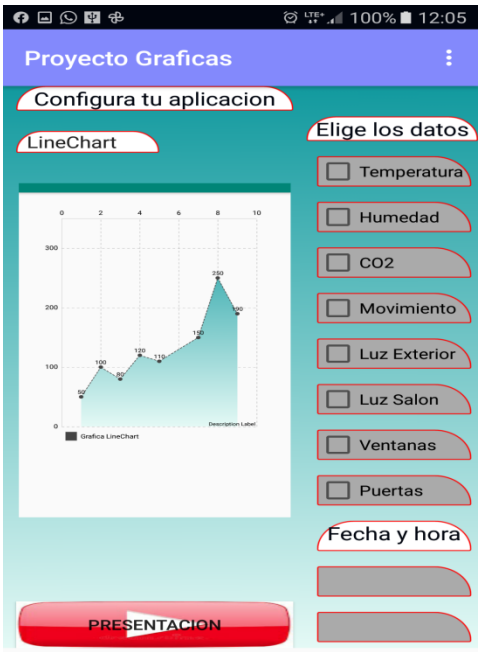
En primer lugar se crea el Intent, indicando la clase de donde se crea, en este caso Login, y la clase donde se dirige, en este caso Configuracion.

Si se necesita enviar información, se utiliza la función putExtra a la cual se le pasa una clave, en este caso `"server"`, y la variable que se necesita enviar, en este caso el String server que ha introducido el usuario.

Después, desde la clase actual (Login) se utiliza la función startActivity pasándole el intent cread (conf)

La clave se debe utilizar en la clase receptora (Configuracion) para recibir el string enviado, es decir el sistema comprueba la coincidencia de la clave, y recibe el dato. Es muy útil la utilización de la clave, dado que hace posible el envió de varios datos, y para distinguirlos se utiliza una clave por cada variable.

5.6.3 Clase Configuracion:



En esta clase se dispone de los botones necesarios para la elección de los datos que se necesitan visualizar (botones checkbox), campo para elegir la fecha, campo para elegir la hora y minuto, un spinner para la elección de la gráfica deseada y el botón de la presentación. Los campos fecha y hora realizan la conexión con las clases DatePickerDialog y TimePickerFragment respectivamente, que devolverán un calendario y un reloj para la elección de la fecha y la hora deseadas desde cuando interesa presentar los datos guardados en la base de datos, es decir los datos requeridos al servidor serán a partir de esta fecha y hora hasta el presente.

Cada vez que el usuario requiere la fecha o la hora, se crea una nueva instancia de estas clases, y se presentantan:

```
/**
 * @param v
 * Para crear una instancia de nuestro datePicker
 * pasándole el EditText debemos usar el método estático que definimos como "newInstance"
 * en lugar de usar el constructor por defecto
 */
public void showDatePickerDialog(EditText v) {
    DialogFragment newFragment = DatePickerDialog.newInstance(v);
    // Mostrar el datePicker
    newFragment.show(getSupportFragmentManager(),  tag: "datePicker");
}

/**
 * @param vi
 * Para crear una instancia del TimePicker
 * pasándole el EditText se debe usar el método estático que se define como "newInstance"
 * en lugar de usar el constructor por defecto
 */
public void showTimePickerFragment(EditText vi){
    DialogFragment newFragment = TimePickerFragment.newInstance(vi);
    // Mostrar el TimePicker
    newFragment.show(getSupportFragmentManager(),  tag: "timePicker");

}
```

En este caso he optado por un evento onClick a través de un switch, con lo cual el usuario elige abrir el calendario o el reloj, llamando a la función correspondiente que abre la presentación de la clase elegida:

```
/**
 * @param view
 * PARA LA ELECCION DE LA FECHA Y HORA
 */
@Override
public void onClick(View view) {
    //Determinamos qué elemento ha sido pulsado
    switch (view.getId()) {
        case R.id.fech:
            //Mostrar el datePicker
            showDatePickerDialog((EditText) view);
            break;
        case R.id.hor:
            // Monstrar el TimePicker
            showTimePickerFragment((EditText) view);
            break;
    }
}
```

A los EdiText encargados de abrir estas clases, se le añaden el evento onClickListener, encargado de escuchar el evento producido cuando el usuario toca la pantalla sobre la fecha o la hora:

```
//Le añadimos un listener para la fecha
EditText inputFecha = (EditText) findViewById(R.id.fech);
inputFecha.setOnClickListener(this);
//Le añadimos un listener para la hora
EditText inputHora = (EditText) findViewById(R.id.hor);
inputHora.setOnClickListener(this);
```

El spinner en el fichero de presentación de la interfaz: activity_configuracion.xml, abarca un RelativeLayout que a su vez tiene un ImageView y un Button para que abra el desplegable para la elección de la gráfica deseada.

Las entradas del desplegable se realizan desde el fichero array_anime.xml del directorio values, es decir toma sus valores (entradas) de este fichero.

```
<Spinner
    android:id="@+id/getIma"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginRight="10dp"
    android:layout_marginLeft="10dp"
    android:background="@drawable/redondear_top"
    android:entries="@array/anime"
    android:prompt="Selecciona una Grafica" />
<RelativeLayout
    android:layout_width="230dp"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="330dp"
        android:layout_marginTop="30dp"
        app:srcCompat="@mipmap/img01" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="46dp"
        android:layout_alignLeft="@+id/imageView"
        android:layout_alignRight="@+id/imageView"
        android:layout_alignParentBottom="true"
```

Array_anime.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="anime">
        <item>LineChart</item>
        <item>BarChart</item>
        <item>LineBarChart</item>

    </string-array>
</resources>
```

Al cambiar de valor del desplegable, se cambiara también la foto de la gráfica que vamos a elegir, de esta forma se realiza una mejor presentación, tanto del diseño de la página como en la elección de la gráfica deseada.

En java, es necesario un array de int que guarda los números enteros que corresponden a cada foto guardadas en el directorio mipmap del proyecto:

```
// Con el sigueinte array se tomara el numero entero que corresponde a cada una de las imagenes
// este se encuentra en gen/R.java, clase generada por el mismo proyecto
int[] imagenes = {R.mipmap.img01, R.mipmap.img02, R.mipmap.img03};
```

Para los eventos que realiza el spinner se utiliza el método `initialize()` que también es encargado de crear los objetos necesarios:

```
/*
 * El siguente metodo es usado para crear los objetos en concretos
 * asi como tambien los eventos que hara el spinner
 */
private void initialize() {
    //busca el imagenview del activity_configuracion.xml
    img = (ImageView) findViewById(R.id.imageView);
    //busca el spinner del activity_configuracion.xml
    cambiar = (Spinner) findViewById(R.id.getIma);
    //OnItemSelectedListener() se ejecuta al hacer clic en el spinner
    cambiar.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener(){

        public void onItemSelected(AdapterView arg0, View arg1, int arg2, long arg3) {
            // TODO Auto-generated method stub
            //por medio de arg2 se obtiene un indice del array_anime.xml
            //y de esa forma lo pasa a img, para recuperar la imagen esperada
            img.setImageResource(imagenes[arg2]);
        }

        public void onNothingSelected(AdapterView arg0) {
            // TODO Auto-generated method stub

        }

    });
```

En esta clase se recibe a través del Intent la dirección del servidor desde la clase Login, utilizando la misma clave que en la clase Login:

```
//Intent para recuperar la direccion del server desde Login
Intent i = this.getIntent();
final String server = i.getStringExtra("server");
```

A través del constructor correspondiente de la clase Aux, se pide al servidor los datos seleccionados por el usuario, pasándole al constructor la fechahora y la ruta al fichero configuración.php.

En este caso, para la construcción de la ruta se utiliza la variable recibida desde la clase Login añadiéndole la ruta al fichero configuración.php

```
String serve;
if(server.isEmpty()||server== null){
    //Se accede a la clase Chart para recuperar la ruta al servidor en
```

```
el caso
    //de que usuario vuelve a esta Activity
  serve=Chart.servidor;
    if(serve.isEmpty()||serve== null){
        //Se accede a la clase BarLineChart para recuperar la ruta al
servidor en el caso
        //de que usuario vuelve a esta Activity
        serve=BarLineChart.servidor;
    }
    ruta = "http://"+serve+"/ProyectoGraficas/configuracion.php";//Se
configura la ruta al servidor
}else {
    ruta = "http://"+server+"/ProyectoGraficas/configuracion.php";//Se
configura la ruta al servidor
}
```

Como se puede observar, se ha generado la ruta a partir de la variable "server" recibida por el intent desde Login, pero también se contempla los casos en los cuales el usuario vuelve de las clases Chart o BarLineChart a esta clase, y de esta manera la variable server desde Login queda en "null".

En estos casos, la ruta se recibe desde estas clases, accediendo directamente a la variable static servidor de estas clases, comprobando que no se queda en "null" Mas adelante explicare como se pasa la ruta a estas clases.

Por igual que en la clase Login, se crea la respuesta Response.Listener con su método sobrescrito onResponse, se le pasa la información al constructor correspondiente de la clase Aux y la petición se añade a la una cola Voley

```
//Llamo al constructor de la clase Auxiliar
Auxiliar r = new Auxiliar(fechahora,ruta,respuesta);
// Utilizo Volley para la comunicacion, Crear nueva cola de peticiones
//Usar versión no oficial de Volley desde Maven Central,obtener el
.jar a través de una regla de de construcción externa en build.gradle
RequestQueue cola = Volley.newRequestQueue(Configuracion.this);
cola.add(r);
```

Se puede observar que en este caso no se espera por parte del constructor la devolución del error, ya que llegados a este punto la ruta al servidor es correcta y también la conexión a internet u otros errores producidos por la conexión.

En método sobrescrito onResponse que trata la respuesta, y se crea el JNSONObject a partir de esta respuesta.

```
//Se crea la nueva respuesta del servidor

Response.Listener<String> respuesta = new Response.Listener<String>(){

    @Override
    public void onResponse(String response) {

        try{
        //Creamos el objeto jsonRespuesta para la respuesta del servidor
        JSONObject jsonRespuesta = new JSONObject(response);


            //Recibimos el parametro boolean succes desde el fichero php del servidor
        boolean ok = jsonRespuesta.getBoolean( name: "success");
```

A través del JNSONObject se recibe la variable success (boolean) por el nombre de esta que se utiliza para el dialogo AlertDialog que indicara si no hay datos disponibles desde la fecha indicada. Esta variable es enviada desde el servidor. En el fichero configuración.php se crea un array donde se inserta esta variable a true o false según el resultado de la consulta sql realizada por el fichero php al servidor SQL.

```
$ar["success"]=false;
// Se inicializa el array as a null por si no hay datos para
$as=null;
while ($fila = $consulta->fetch_assoc()) {
      $i++;
      //Se pone a true por recojer el valor booleano en java
      $ar["success"]=true;
      //Se añaden los campos a cada array
```

AlertDialog.Builder de la clase Configuracion

```
}else{
    //Se crea la alerta
    AlertDialog.Builder alerta = new AlertDialog.Builder( context: Configuracion.this);
    alerta.setMessage("No existen datos desde esta fecha ")
            .setNegativeButton( text: "Reintentar",  listener: null)
            .create()
            .show();
```

Si la variable es false, se devuelve a la interfaz de la misma clase (Configuracion), y en caso contrario, se procede a rellenar con los datos desde el servidor, un array de doble dimensión de Strings creado anteriormente:

```
 //Se crea un arrayList de doble dimension con los datos que se van a recuperar
final ArrayList<List<String>> datosServidor = new ArrayList<~>();
 //Se crea la primera sublista
 datosServidor.add(new ArrayList<String>());
```

Se ha creado este array de doble dimensión, para colocar ordenadamente los datos recibidos desde el servidor, facilitando de esta forma la lectura y presentación de estos datos.

Para el relleno de cada sub lista, se utiliza un array de Strings con los nombres de los parámetros esperados, un array de los checkbox y un contador.

```
//Array con los parámetros en string necesarios para la recuperación
de los datos enviados desde el servidor
private String parametros[]={"temperatura","humedad","nivelCO2",
"movimiento","luzExterior","luzSalon","ventanas","puertas"};

//ArrayList para guardar los CheckBox
 ArrayList<CheckBox> checkbox =new ArrayList<>();

//Componentes de la interfaz
private CheckBox
temperatura,humedad,codos,movimiento,luzExterior,luzSalon,ventanas,pue
rtas;

temperatura= (CheckBox) findViewById(R.id.temp);
checkbox.add(temperatura);
humedad= (CheckBox) findViewById(R.id.hum);
checkbox.add(humedad);
codos= (CheckBox) findViewById(R.id.codos);
checkbox.add(codos);
movimiento= (CheckBox) findViewById(R.id.movim);
checkbox.add(movimiento);
luzExterior= (CheckBox) findViewById(R.id.exterior);
checkbox.add(luzExterior);
luzSalon= (CheckBox) findViewById(R.id.salon);
checkbox.add(luzSalon);
ventanas= (CheckBox) findViewById(R.id.ventanas);
checkbox.add(ventanas);
puertas= (CheckBox) findViewById(R.id.puertas);
checkbox.add(puertas);


//Contador que representa el numero de sublistas que se creara:
contadorIsChecked+2
int contadorSubListas=0;
```

En los bucles for se añade a cada sublista los datos ordenados, si han sido seleccionados por el usuario; si los checkbox han sido seleccionados, se crea una nueva sublista para cada uno, donde se almacena el valor de la base de datos desde la fecha seleccionada.

```
//Se añade al ArrayList datosServidor los valores recibidos, dependiendo de las casillas seleccionadas
//Se creara tantas sublistas los que se necesitan, segun las casillas seleccionadas
for (int i=0;i<=7;i++){
    if(checkbox.get(i).isChecked()){
        //Se añade una nueva sublista por cada checkbox seleccionado
        datosServidor.add(new ArrayList<String>());

        //bucle que crea el parametro con el nombre necesario para recibir el dato del servidor
        //y lo guarda en su pozicion exacta en la sublista
        for (int y=1;y<=numeroDeIndicesSublistas;y++) {
            parm = parametros[i] + y;
            resultParm = jsonRespuesta.getString(parm);
            datosServidor.get(contadorSubListas).add(resultParm);
        }
        contadorSubListas++;
    }
}
```

A la variable "parm" se le concede el valor correspondiente del array parámetros concatenando un número (y) del bucle for. Este bucle for, se ejecuta tantas veces cuantas filas hay en la base de datos correspondientes a la consulta realizada con la fecha y hora introducida.

La variable numeroDeIndicesSublista es devuelta por el fichero configuración.php una vez realizada la consulta y recuperada en el método onResponse:

```
35      $ar["num"]=$i;
36      //Se concatenan los arrays en el array as
37      //Por cada bucle se cambia el valor del $ar
38      $as=array_merge($ar,$date,$temp,$hum,$codos,$mov,$luzext,$luzsalon,$
39
40  }
41   if($as==null){
42   $as=$ar;
43   }
```

```
//El numero de filas encontradas encontradas en la BBDD;
// representa el numero de datos que se almacenara en cada sublista
String numDeIndicesSublistas=jsonRespuesta.getString( name: "num");
int numeroDeIndicesSublistas=Integer. parseInt(numDeIndicesSublistas);
```

De esta forma cada nombre esperado desde el servidor va a tener el nombre del parámetro concadenado con el número de veces que aparece en la consulta del fichero configuracion.php

El String devuelto por este fichero es en formato json_encode:

```
44 //Se responde en formato json
45
46  echo json_encode($as);
47
48 ?>
```

{"success":true,"num":5,"fecha1":"2020-07-21 20:53:20","fecha2":"2020-07-21 20:54:43","fec
18:31:49","temperatura1":"21","temperatura2":"21","temperatura3":"23","temperatura4":"25",

Se puede observar que, cada campo tiene como nombre el mismo campo y un número que representa el número de fila correspondiente de donde ha sido tomado. De aquí la necesidad de poner el nombre con su número de cada campo, al dato esperado desde el servidor.

Este array de doble dimensión se enviara a la clase correspondiente para la presentación de las gráficas a través de un Intent dentro de un bucle for

```
// Se pasa a la clase corespondiente los datos obtenidos del servidor
for (int i=0;i<=contadorSubListas-1;i++){
    String ar="ar"+i;
    ArrayList pasar=new ArrayList(datosServidor.get(i));
    in.putExtra(ar, pasar);
```

Se utiliza un bucle para pasar cada sablista a siguiente clase, ya que el Intent no permite el paso de un array de doble dimensión. En la siguiente clase se recupera las sublistas, según la clave “ar”+i

En la primera de las sublistas del array de doble dimensión datosServidor, se almacena el `numDeIndicesSublistas` recibido desde el servidor, el nombre de la gráfica seleccionada por el usuario y el número de sublistas creadas

```
//Se añade en la primera pozicion de la primera sublista el numero de
filas(numeroDeIndicesSublistas) encontradas en la BBDD
//Se añade como string ya que las listas acceptan solamente strings
// En esta pozicion el contadorListas=0 que representa la primera
sublista
datosServidor.get(contadorSubListas).add(numDeIndicesSublistas);


//Se añade el nombre de la grafica seleccionada en la segunda pozicion
de la primera sublista
//Se recupra el nombre de la grafica elegida para presentar y se
guarda en la primera sublista
String grafica=cambiar.getSelectedItem().toString();
datosServidor.get(contadorSubListas).add(grafica);
```

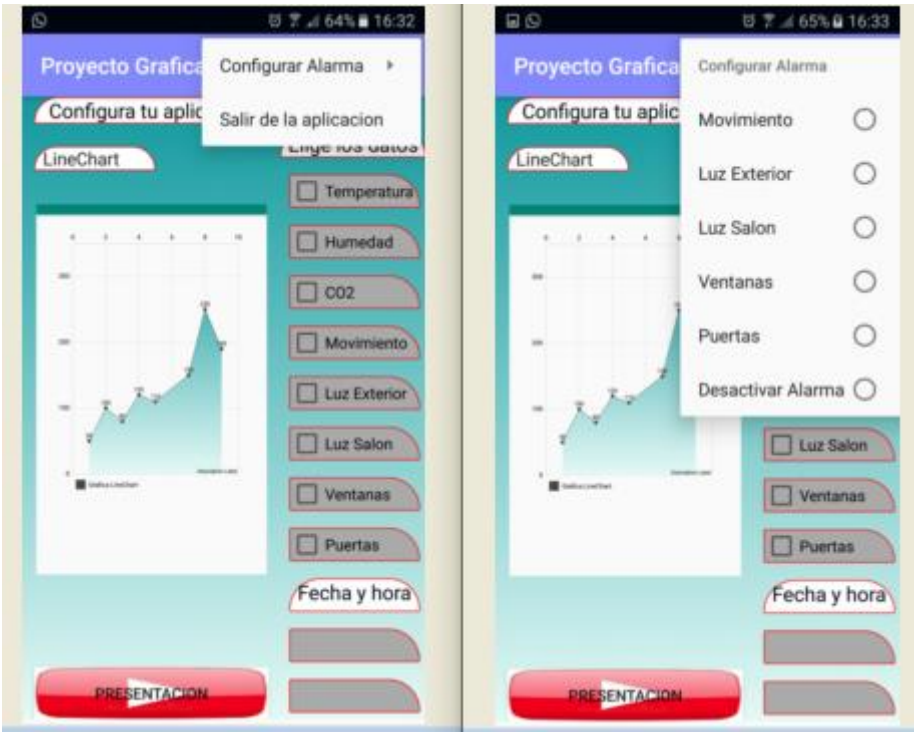
```
// el numero total de sublistas creadas en el array de doble dimension
datosServidor
String numSublistas= Integer.toString(contadorSubListas);
datosServidor.get(0).add(numSublistas);
```

En el mismo bloque try/catch se prepara un nuevo Intent, según la gráfica seleccionada, comprobando esta en el array `datosServidor` y se envía a la clase seleccionada a través de putExtra el contador `contSbuList`, el array de posiciones **`pozicionCheckbox`** y la dirección del servidor.

Después se inicia la clase correspondiente, sin apagar la clase Configuración, dado que el usuario podría necesitar volver a esta clase para cambiar de ajustes. Se pone a 0 el **`contadorSubListas`** y **`pozicionCheckbox`** por la misma razón.

```
//Atraves del intent pasa la informacion a la clase elegida segun el
promt de los dibujos de graficas
Intent in;

if(datosServidor.get(0).get(1).equals("LineChart")||datosServidor.get(
0).get(1).equals("BarChart")) {
    in = new Intent(Configuracion.this, Chart.class);
}else {
    in = new Intent(Configuracion.this, BarLineChart.class);
}
//Se pasa de primero el numero de filas sleccionadas de la BBDD a
travez de la fecha y hora elegida
String contSbuList=Integer.toString(contadorSubListas);
in.putExtra("sublistas",contSbuList);
//Se pasa el array de poziciones
in.putExtra("pozicion", pozicionCheckbox);

//Se pasa la direccion del servidor
in.putExtra("servidor", server);

Configuracion.this.startActivity(in);
//Se pone el contador a 0 por si el usuario vuelve a esta Activity
contadorSubListas=0;
//Se vasea por si el usuario vuelve a esta Activity
pozicionCheckbox.clear();
```

Menú de la clase Configuración



En la misma clase Configuración he creado y definido el menú que tiene un desplegable para la elección de la alarma y un botón para apagar la aplicación.

En el desplegable se presentan los campos que se pueden elegir para la alarma, siendo permitido un solo campo a seleccionar, es decir si se selecciona otro, el anterior pierde el foco.

En la clase Configuracion se han definido los métodos del menú: `onCreateOptionsMenu(Menu menu)` y `onOptionsItemSelected(MenuItem item)`

En este último utiliza un bloque try/catch (para las excepciones) donde se vuelve a recuperar la ruta al servidor, esta vez añadiendo la ruta al fichero consulta.php, que devuelve desde su consulta al servidor SQL los valores seleccionados por la última fecha de la base de datos

consulta.php

$consulta = $con->query("SELECT * FROM `datos` WHERE `date` = (SELECT MAX(`date`) from `datos`)");

```
String serve,rutaAlarma;
if(server.isEmpty()||server== null){
    //Se accede a la clase Chart para recuperar la ruta al servidor en
el caso
    //de que usuario vuelve a esta Activity
    serve=Chart.servidor;
```

```
    if(serve.isEmpty()||serve== null){
        //Se accede a la clase BarLineChart para recuperar la ruta al
servidor en el caso
        //de que usuario vuelve a esta Activity
        serve=BarLineChart.servidor;
    }
    rutaAlarma = "http://"+serve+"/ProyectoGraficas/consulta.php";//Se
configura la ruta al servidor
}else {
    rutaAlarma =
"http://"+server+"/ProyectoGraficas/consulta.php";//Se configura la
ruta al servidor
}
```

A través de un switch, se selecciona la opción seleccionada por el usuario en los botones del menú, y comprobando por el título del ítem seleccionado, se establece el valor de la variable Static tipo, y se llaman a las funciones apagar y establecer alarma.

```
switch (item.getItemId()) {
 //Se utiliza el el titulo de cada item del menu para establecer que
buton ha sido seleccionado
    case R.id.Opcion2:
        //Varias formas de cerrar la actividad
        //System.exit(0);
        finishAffinity();
        //android.os.Process.killProcess(android.os.Process.myPid());
    case R.id.item_movimiendo:
        titulo = (String) item.getTitle();
        if (titulo.equals("Movimiento")){
        tipo="movimiento";
        try {
            apagarAlarma();
            establecerAlarma();

        } catch (Exception e) {
            displayExceptionMessage("  Exception MOVIMIENTO " +
e.getMessage());
            e.printStackTrace();
        }}
    case R.id.item_exterior:
         titulo = (String) item.getTitle();
        if (titulo.equals("Luz Exterior")){
            tipo="luzExterior";

            try {
                apagarAlarma();
                establecerAlarma();
```

```
            } catch (Exception e) {
                displayExceptionMessage("  Exception luzExterior " +
e.getMessage());
                e.printStackTrace();
            }}

    case R.id.item_salon:
        titulo = (String) item.getTitle();
        if (titulo.equals("Luz Salon")){
            tipo="luzSalon";

            try {
                apagarAlarma();
                establecerAlarma();

            } catch (Exception e) {
                displayExceptionMessage("  Exception luzSalon " +
e.getMessage());
                e.printStackTrace();
            }}

    case R.id.item_ventanas:
        titulo = (String) item.getTitle();
        if (titulo.equals("Ventanas")){
            tipo="ventanas";
        try {
            apagarAlarma();
            establecerAlarma();
        } catch (Exception e) {
            displayExceptionMessage("  Exception ventanas " +
e.getMessage());
            e.printStackTrace();
        }}
    case R.id.item_puertas:

        titulo = (String) item.getTitle();
        if (titulo.equals("Puertas")){
            tipo="puertas";
        try {
            apagarAlarma();
            establecerAlarma();

        } catch (Exception e) {
            displayExceptionMessage("  Exception puertas " +
e.getMessage());
            e.printStackTrace();
```

```
        }}
    case R.id.desactivar:
        //Mantiene el estado visible del elemento seleccionado en el
menu
        if (item.isChecked()) item.setChecked(false);
        else item.setChecked(true);

        titulo = (String) item.getTitle();
        if (titulo.equals("Desactivar Alarma")){
            try {
                apagarAlarma();
                displayExceptionMessage("Alarma Apagada en class
Configuracion ");

            } catch (Exception e) {
                displayExceptionMessage("  Exception puertas " +
e.getMessage());
                e.printStackTrace();
            }}
        return true;
    default:
        return super.onOptionsItemSelected(item);
}
```

En este mismo bloque, la primera opción se utiliza para salir de la aplicación y la última para parar la alarma y mantener el foco sobre la opción pulsada.

Función establecerAlarma()

En esta función se utiliza un bloque try/catch para las excepciones, se presenta un aviso a través del Toast, se crea un manager de la clase AlarmManager del sistema, un Intent para inciar la clase MyReceiver a través de lo cual se envían las variables tipo establecida en el switch del menú y ru (ruta al fichero **consulta.php**), y un PedingIntent que es la intención pendiente de arrancar la clase MyReceiver.

La línea:

```
pIntent = PendingIntent.getBroadcast(this, 1, intent,
PendingIntent.FLAG_CANCEL_CURRENT);
```

Crea un intent pendiente que se envía por broadcast y va ser recibido en el método onRecive de la clase MyReceiver; “this” indica que se arranca desde este contexto, “1” es un código request necesario para el envió, se añade el intent creado anteriormente y después se indica que si la intención pendiente ya existe, la cancela antes de crear otra nueva (***FLAG_CANCEL_CURRENT)***

La clase del sistema AlarmManager permite acceder a la alarma del sistema. Con la ayuda de Android AlarmManager en Android, se puede programar la aplicación para que se ejecute en un momento específico en el futuro, en nuestro caso cada 60 segudos.

El AlarmManager de Android tiene un bloqueo de activación de la CPU que garantiza que no se suspenderá el teléfono hasta que se maneje la transmisión.

En este proyecto, se utiliza para las repeticiones del PedingIntent, empezando desde `System.currentTimeMillis()`(los milisegundos actuales del sistema) a un intervalo de 60 segundos, pasándole el *pIntent*

```
public  void establecerAlarma(){
    try {
        Toast.makeText(this, "Alarma activada",
Toast.LENGTH_SHORT).show();
         manager = (AlarmManager)
this.getSystemService(Context.ALARM_SERVICE);

        Intent intent = new Intent(this, MyReceiver.class);
        intent.putExtra("movim", tipo);
        intent.putExtra("ruta", ru);
        //token que permite a MyReceiver.class usar los permisos de la
aplicación para ejecutar un código predefinido
        //PendingIntent.getBroadcast inicia el BroadcastReceiver
        pIntent = PendingIntent.getBroadcast(this, 1, intent,
PendingIntent.FLAG_CANCEL_CURRENT);
        //Repite el intent impezando desde tiempo actual a intervalos
de 60 segundos
        manager.setRepeating(AlarmManager.RTC_WAKEUP,
System.currentTimeMillis() ,1000 * 60 * 1, pIntent);

    }catch (Exception e){
        Toast.makeText(this,e.getMessage(),Toast.LENGTH_SHORT).show();
    }
```

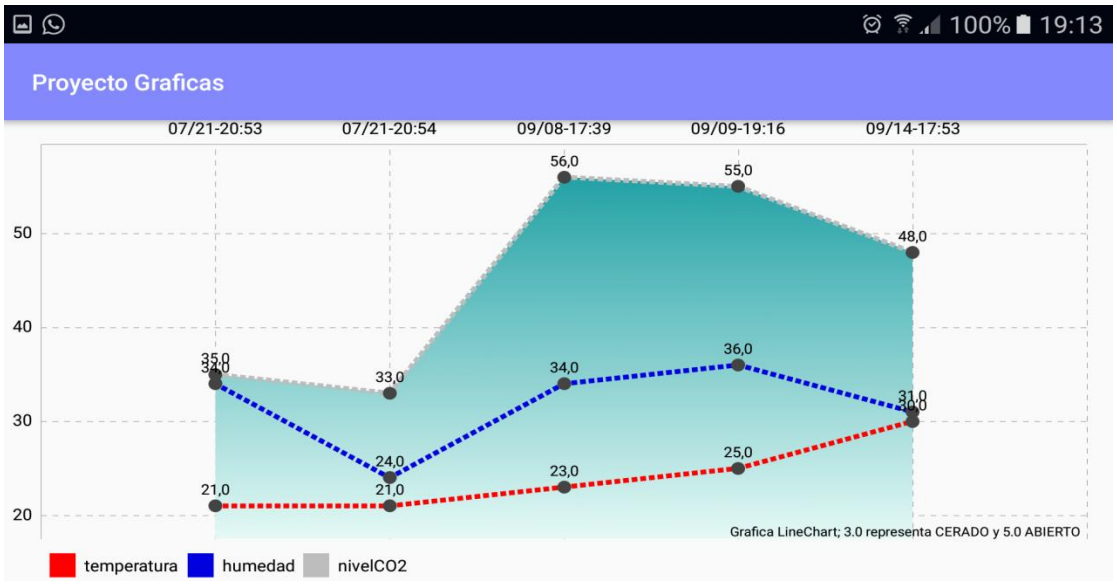
Función apagarAlarma()

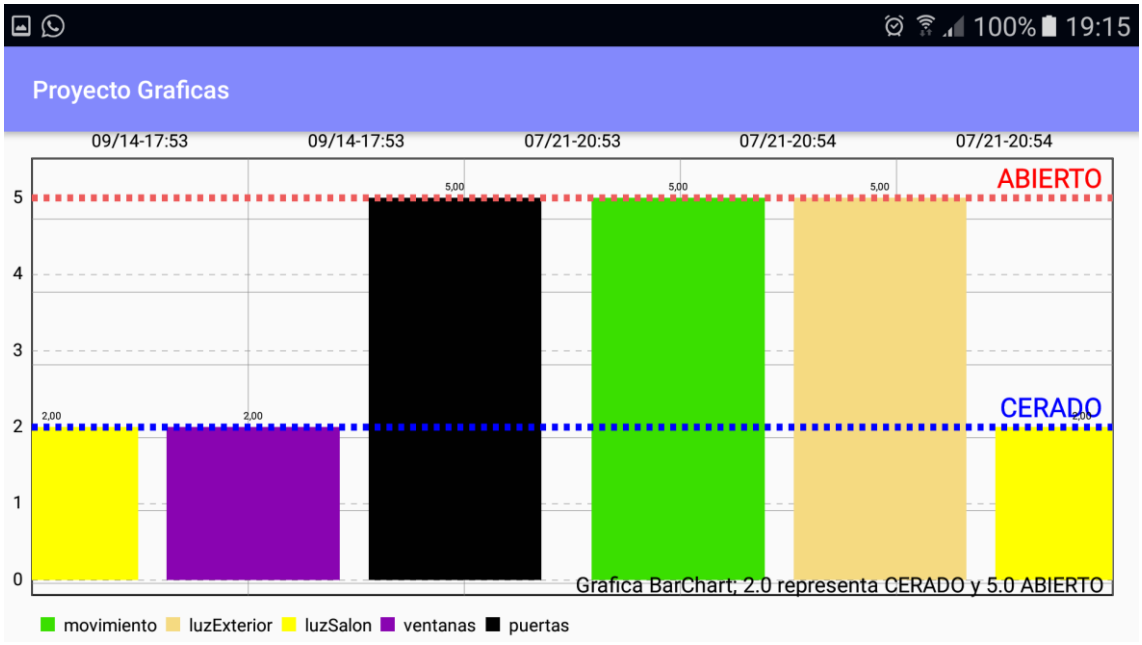
En esta función se comprueba si el manager de la clase AlarmManager es diferente de null (esta creado) y se cancela el PedingIntent, y se para el servicio del segundo plano AuxAlarma.class.

En caso contrario, solamente se para el servicio del segundo plano. AuxAlarma.

```
public  void apagarAlarma(){

    if (manager!= null) {

        manager.cancel(pIntent);

        stopService(new Intent(Configuracion.this, AuxAlarma.class));

        //Prueba de funcionalidad

        //displayExceptionMessage("Alarma Apagada en class
Configuracion ");

    }else{
        stopService(new Intent(Configuracion.this, AuxAlarma.class));

        //Prueba de funcionalidad

        //displayExceptionMessage("Servicio Parado en class
Configuracion ");

    }
```

5.6.4 Clase Chart:





Esta clase es arrancada por el Intent de la clase Configuracion y presenta una de las 2 graficas de arriba, según la elección realizada por el usuario a través del spinner.

Se crea 2 arrays para la presentación:

```
//Se crea adicional 2 arrays con los nombres todas las columnas y con los colores de cada una
String parametros[]={"temperatura","humedad","nivelCO2","movimiento","luzExterior","luzSalon","
int[] col = {Color.rgb( red: 255,  green: 0,  blue: 0), Color.rgb( red: 1,  green: 1,  blue: 223), Col
        , Color.rgb( red: 58,  green: 223,  blue: 0) , Color.rgb( red: 245,  green: 218,  blue: 129) ,
        Color.rgb( red: 137,  green: 4,  blue: 177), Color.rgb( red: 1,  green: 1,  blue: 1)};
```

En el método onCreate se crean las variables linechart y barchart que representan las interfaces del fichero activity_chart.xml, se crea un Intent para recuperar las variables `numeroDeSublistas`, `contadorSubListas`, *`servidor`*, `ArrayList<Integer> posición`, se crea un array de doble dimensión `ArrayList<Integer> posición` para recuperar las sublistas enviadas, y en el bucle for se almacenan estas sublistas en este array en formato ArrayList.

```
for (int i = 0; i <= contadorSubListas - 1; i++) {


    //Se crea tantas sublistas, lo que se espera desde la Activity
Configuracion y se guarda cada una en el ArrayList datosS


    String ar = "ar" + i;
```

```
    ArrayList lista = (ArrayList<String>)
getIntent().getSerializableExtra(ar);

    datosS.add(lista);

}
```

Con la siguiente comprobación se arranca la interfaz deseada, llamando la función indicada:

```
if(datosS.get(0).get(1).equals("LineChart")) {

    // Evita presentar el mensage que el barchart no tiene datos

    barchart.setNoDataText("");

    //Para el LineChart

    xAxisDataLineChart(datosS, pozicion);

}else  {

    // Evita presentar el mensage que el linechart no tiene datos

    linechart.setNoDataText("");

    //Para el BarChart

    xAxisDataBarChart(datosS, pozicion);

}
```

La función setNoDataText, evita la presentación por defecto del informe por parte del sistema que la gráfica no tiene datos, ya que presentando solamente una gráfica, la otra va ser vacía.

Función xAxisDataLineChart

Recibe el array de doble dimensión datos, y el ArrayList posición y configura la axis X de la gráfica LineChart y la axis Y izquierda. Es posible presentar los datos también en el lado derecho, pero no es el caso en este proyecto. También se establece el grid de la grfica y la colocación de las fechas en la parte superior. Por lo último se llama a la función setDataLineChart

```
public void xAxisDataLineChart(ArrayList<List<String>>
datosS,ArrayList<Integer> pozicion ) {

    //Array String de las fechas recibidas en la funcion
getXAxisValues()

    final String[] labels = getXAxisValues(datosS);

    XAxis xAxisLine = linechart.getXAxis();

    //Las lineas orizontales del grid de la grafica con las
dimensiones establecidas

    xAxisLine.enableGridDashedLine(10f, 10f, 0f);

    //Dimension orizontal del grid

    xAxisLine.setAxisMaximum(10f);

    xAxisLine.setAxisMinimum(0f);

    //xAxisLine.setCenterAxisLabels(true);

    xAxisLine.setDrawLabels(true);

    xAxisLine.setDrawAxisLine(true);

    //Añade a la axisX el array String de las fechas recibidas en la
funcion getXAxisValues()

    xAxisLine.setValueFormatter(new IndexAxisValueFormatter(labels));
```

```
 YAxis leftAxisLine =linechart.getAxisLeft();

    leftAxisLine.removeAllLimitLines();

    //Las lineas verticales del grid de la grafica con las dimensiones
establecidas

    leftAxisLine.enableGridDashedLine(10f, 10f, 0f);

    leftAxisLine.setDrawZeroLine(false);

    leftAxisLine.setDrawLimitLinesBehindData(false);

    linechart.getAxisRight().setEnabled(false);

    setDataLineChart(datosS,pozicion);

}
```

Función setDataLineChart

Esta función tiene 2 bloques try/catch, en el primero se extraen los datos desde el array de doble dimensión y se colocan en las variables valor de tipo Entries utilizadas en este tipo de gráficas. Estas variables se agrupan en un nuevo arraylist "datos".

La variable "leyenda" de tipo LineDataSet, utiliza este array para configurar la presentación de estos datos, junto con el array "parámetros" y "posición". Además aquí se configura la misma leyenda de la gráfica donde se presentara el código de colores utilizado para cada dato. Para el color de fondo se utiliza el fichero fade_blue.xml del directorio drawable.

En el segundo bloque tray/chart se dibuja la gráfica con algunos ajustes más.

```
private void setDataLineChart(ArrayList<List<String>>
datosS,ArrayList<Integer> pozicion ) {

    //Se crea el Array para la axix Y

    ArrayList<ILineDataSet> yAxix = new ArrayList();

    //Se crea la leyenda LineDataSet
```

```
    LineDataSet leyenda=null;

try{
        //Se saca el número de sublistas (cuantos checkbox
seleccionados +2 hay) para el bucle
        String numDeListas =datosS.get(0).get(2);
        int numeroDeListas=Integer.parseInt(numDeListas);

        //Se saca el número de filas sleccionadas de la BBDD a travez
de la fecha y hora elegida  para el bucle
        //Reprezenta tambien el número de datos almacenados en cada
sublista
        String numDeFilas =datosS.get(0).get(0);
        int numeroDeFilas=Integer.parseInt(numDeFilas);
        //Prueba de funcionalidad
        //displayExceptionMessage("LISTAS "+numeroDeListas+" INDICES
SUBLISTAS "+numeroDeFilas);
        //Interesa empezar por la tercera sublista ya que en la
primera (poz.0 del array)
        // se almacena el número de filas y columnas de la BBDD
  //I en la segunda (poz.1 del array) las fechas
        int contador=0;
        int cont=0;
        for (int i=2;i<=numeroDeListas-1;i++){
            //El contador representa la pozicion en Axis X
            contador=0;
//Por cada bucle se pone el contador a 0
//Se crean tantos ArrayList cuatos tipos de datos seleccionados con
los checkbox hay
            ArrayList datos = new ArrayList();
            //Bucle para extraer los datos de cada sublista
            for (int y=0;y<=numeroDeFilas-1;y++) {
```

```
                //Prueba de funcionalidad

  //displayExceptionMessage("DATOS FILA "+i+" POZICION "+y+" :
"+datosS.get(i).get(y));

                String dato =datosS.get(i).get(y);

                int datoInteger=Integer.parseInt(dato);

                // Se cambia el valor para una mejor presentacion

                if(datoInteger==0){

                    datoInteger=3;

                }

    // Se cambia el valor para una mejor presentación

                if(datoInteger==1){

                    datoInteger=5;

      //Entry valor=new Entry(Float.valueOf(1+"ABIERTO"), contador);

                }

                Entry valor = new Entry(contador, datoInteger);

        //Se añaden los datos al ArrayList

                datos.add(valor);
                  contador++; }

             //Linea de datos

            //Se añade los datos y la descripcion del color

            leyenda = new LineDataSet(datos,parametros[pozicion.get
(cont)]);

            //Color de la línea

            leyenda.setColor(col[pozicion.get(cont)]);

            //Línea de datos dibujo

            leyenda.enableDashedLine(10f, 5f, 2f);

            //Línea de datos grueso

            leyenda.setLineWidth(3f);

            //Puntos de las líneas
```

```
            leyenda.setCircleColor(Color.DKGRAY);

            leyenda.setCircleRadius(4f);

            //Los puntos están rellenos

            leyenda.setDrawCircleHole(false);

            //Texto

            leyenda.setValueTextSize(9f);

            //Habilita el fondo de color por debajo de las líneas

            leyenda.setDrawFilled(true);

            leyenda.setFormLineWidth(1f);

            leyenda.setFormLineDashEffect(new DashPathEffect(new
float[]{10f, 5f}, 0f));

            leyenda.setFormSize(15.f);


//Color de fondo en degradable soportado a partir del API level 18

            if (Utils.getSDKInt() >= 18) {
 Drawable drawable = ContextCompat.getDrawable(this,
R.drawable.fade_blue);

leyenda.setFillDrawable(drawable);

            } else {

  leyenda.setFillColor(Color.DKGRAY);  }

            yAxix.add(leyenda);

            cont++;

        }

    }catch (Exception e){
     displayExceptionMessage("Exception LineData CLASS:CHART :
"+e.getMessage());

        e.getMessage(); //Tratamos las posibles excepciones

    }

    LineData datosLineChart = new LineData(yAxix);

    try{
```

```
        linechart.animateXY(2000, 2000);

        //Se dibuja el mChart

        linechart.setData(datosLineChart);

        Description description = new Description();

        description.setText("Grafica LineChart; 3.0 representa CERADO
y 5.0 ABIERTO ");

        linechart.setDescription(description);

        //Dimension minimo y maximo, dibuja tantas lineas verticales
cuantas fechas hay disponibles

        linechart.getXAxis().setAxisMinimum(-1);


linechart.getXAxis().setAxisMaximum(Integer.parseInt(datosS.get(0).get
(0)));


 //Coloca las fechas desde la primera linea vertical del grid



linechart.getXAxis().setLabelCount(Integer.parseInt(datosS.get(0).get(
0))+2, true);

        if(Integer.parseInt(datosS.get(0).get(0))==1){

            linechart.setNoDataText("ADVERTENCIA: Esta grafica no
representa correctamente los datos debido a que hay solamente una
fecha");

        }

        linechart.invalidate();// refrescar el linechart

    }catch (Exception e){

        displayExceptionMessage("Excepcion linechart CLASS:CHART :
"+e.getMessage());

    }

}
```

Función_xAxisDataBarChart

Al igual que en la función xAxisDataLineChart, se configuran las axis X e Y, esta vez para la gráfica BarChart, con las variables necesarias para este tipo de grafica

```
public void xAxisDataBarChart(final ArrayList<List<String>> datosS,
ArrayList<Integer> pozicion ) {

    //BLOQUE ENCARGADO DE CONFIGURAR EL XAXIS
    try{
        final String[] labels = getXAxisValues(datosS);
        //Prueba de funcionalidad
        //displayExceptionMessage("labels size "+labels.length);
        XAxis xAxisBar = barchart.getXAxis();
        //xAxisBar.setDrawGridLines(true);
        xAxisBar.setDrawAxisLine(true);
        xAxisBar.setDrawLabels(true);
        //xAxisBar.setAxisMaximum(1);

        //Coloca en el centro de los recuadros las fechas, a partir de
la linea con el valor

        xAxisBar.setCenterAxisLabels(true);


//xAxisBar.setLabelCount(Integer.parseInt(datosS.get(0).get(0)));

        xAxisBar.setPosition(XAxis.XAxisPosition.TOP);

        //Añade a la axisX el array String de las fechas recibidas en
la funcion getXAxisValues()

        xAxisBar.setValueFormatter(new IAxisValueFormatter() {

            @Override
```

```
            public String getFormattedValue(float value, AxisBase
axis) {

                return labels[(int) value % labels.length];

            }

        });
        LimitLine ll1 = new LimitLine(5f, "ABIERTO");

        ll1.setLineWidth(4f);

        ll1.enableDashedLine(10f, 10f, 0f);

        ll1.setLabelPosition(LimitLine.LimitLabelPosition.RIGHT_TOP);

        ll1.setTextSize(15f);

        ll1.setTextColor(RED);

        LimitLine ll2 = new LimitLine(2f, "CERADO");

        ll2.setLineWidth(4f);

        ll2.enableDashedLine(10f, 10f, 0f);

        ll2.setLabelPosition(LimitLine.LimitLabelPosition.RIGHT_TOP);

        ll2.setTextSize(15f);

        ll2.setTextColor(BLUE);

        ll2.setLineColor(BLUE);
```

```
        YAxis leftAxis = barchart.getAxisLeft();

        leftAxis.removeAllLimitLines();

        leftAxis.addLimitLine(ll1);

        leftAxis.addLimitLine(ll2);

        leftAxis.setAxisMinimum(-0.2f);

        leftAxis.enableGridDashedLine(10f, 10f, 0f);

        leftAxis.setDrawZeroLine(false);

        leftAxis.setDrawLimitLinesBehindData(false);

        //Quita los valores presentados en yAxis de la derecha

        YAxis rightAxis = barchart.getAxisRight();

        rightAxis.removeAllLimitLines();

        rightAxis.setDrawLabels(false);

        //Llamo a ejecutar la funcion setDataBarChart()

        setDataBarChart(datosS,pozicion);

    }catch (Exception x){

        displayExceptionMessage("Exception xAxisDataBarChart
CLASS:Chart: "+x.getMessage());

    }

}
```

Función setDataBarChart

Al igual que en la función setDataLineChart, se utilizan los 2 bloqyes try/cach para definir las variables “valor” de tipo BarEntrie agrupadas en el array datos, variable “leyenda” y la presentación del barchart. De remarcar que en este caso se ha configurado un scroolbar para la presentación de los datos, en la línea:

```
//para el scroolbar, presenta maximo 3 barras por fecha

barchart.setVisibleXRangeMaximum(3);

private void setDataBarChart(ArrayList<List<String>>
datosS,ArrayList<Integer> pozicion ) {

    //Se crea el Array para la axix Y

    ArrayList<IBarDataSet> yAxix = new ArrayList();

    //Se crea la leyenda BarDataSet


    BarDataSet leyenda=null;


    //ESTE BLOQUE SE ENCARGA DE EXTRAER LA INFORMACION DEL ARRAYLIST Y
COLOCARLA EN EL BARDATASET (VALORES YAXIS)


    try{


        //Se saca el numero de sublistas (cuantos checkbox
seleccionados +2 hay) para el bucle


        String numDeListas =datosS.get(0).get(2);


        int numeroDeListas=Integer.parseInt(numDeListas);


        //Se saca el numero de filas sleccionadas de la BBDD a travez
de la fecha y hora elegida  para el bucle


        //Reprezenta tambien el numero de datos almacenados en cada
sublista
        String numDeFilas =datosS.get(0).get(0);


        int numeroDeFilas=Integer.parseInt(numDeFilas);
```

```
        //Prueba de funcionalidad

        //displayExceptionMessage("LISTAS "+numeroDeListas+" INDICES
SUBLISTAS (numeroDeFilas) "+numeroDeFilas);


       //Interesa empezar por la tercera sublista ya que en la primera
(poz.0 del array)


        // se almacena el numero de filas y columnas de la BBDD

        //I en la segunda (poz.1 del array) las fechas

        int contador=0;

        int cont=0;

        for (int i=2;i<=numeroDeListas-1;i++) {

            //El contador representa la pozicion en Axis X

            contador = 0;//Por cada bucle se pone el contador a 0

            //Se crean tantos ArrayList cuatos tipos de datos
seleccionados con los checkbox hay

            ArrayList datos = new ArrayList();

            try{

                //Bucle para extraer los datos de cada sublista


                for (int y = 0; y <= numeroDeFilas - 1; y++) {


                    //Prueba de funcionalidad


                    //displayExceptionMessage("DATOS FILA "+i+"
POZICION "+y+" : "+datosS.get(i).get(y));


                    String dato = datosS.get(i).get(y);


                    int datoInteger = Integer.parseInt(dato);


                    // Se cambia el valor para una mejor presentacion
```

```
                    if (datoInteger == 0) {

                        datoInteger = 2;

                    }
                    // Se cambia el valor para una mejor presentacion

                    if (datoInteger == 1) {

                       datoInteger = 5;

                    }

                    BarEntry valor = new BarEntry(contador,
datoInteger);

                    //Se añaden los datos al ArrayList

                    datos.add(valor);

                    contador++;

                }

            }catch(Exception ex){

                displayExceptionMessage("Exception bucle for :
"+ex.getMessage());

                ex.getMessage(); //Tratamos las posibles excepciones

            }

            //Se añade por cada linea la descripccion de la columna y
el color corespondiente
```

```
            leyenda = new BarDataSet(datos,
parametros[pozicion.get(cont)]);

            leyenda.setColor(col[pozicion.get(cont)]);

            yAxix.add(leyenda);

            cont++;

        }

        //Prueba de funcionalidad

        //displayExceptionMessage("CONTADOR "+contador+" CONT "+cont);

    }catch (Exception e){

        displayExceptionMessage("Exception setDataBarChart CLASS:CHART
: "+e.getMessage());

        e.getMessage(); //Tratamos las posibles excepciones

    }

    //BLOQUE ENCARGADO DE CONFIGURAR EL BARCHART PARA DEPUES PASARLO
AL XML

     try {

         BarData dataBarChart = new BarData(yAxix);

         float groupSpace = 0.06f;//espacio entre los grupos de baras

         float barSpace = 0.08f; // espaccio entre baras
```

```
            float barWidth = 0.48f; // Grueso de las baras

            //Configura el BarData con el grueso de las barras
establecido

            dataBarChart.setBarWidth(barWidth);

            barchart.setData(dataBarChart);

            //Dibuja las columnas del grid

barchart.getXAxis().setLabelCount(Integer.parseInt(datosS.get(0).get(0
))+1, true);

            //Se controla la grafica en funcion del numero de datos
seleccionados
            if(Integer.parseInt(datosS.get(0).get(2))==3){
                barWidth = 0.25f; // Grueso de las baras
                dataBarChart.setBarWidth(barWidth);
                // Integer.parseInt(datosS.get(0).get(0)) es el numero de
entradas
                //Presenta el número de fechas encontradas colocandolas
en el TOP


barchart.getXAxis().setAxisMaximum(Integer.parseInt(datosS.get(0).get(
0)));
            }else {
                barchart.groupBars(0, groupSpace, barSpace);

                // Integer.parseInt(datosS.get(0).get(0)) es el número de
entradas
                //Presenta el número de fechas encontradas colocandolas
en el TOP
barchart.getXAxis().setAxisMaximum(0+barchart.getBarData().getGroupWid
th(groupSpace, barSpace) * (Integer.parseInt(datosS.get(0).get(0))));
```

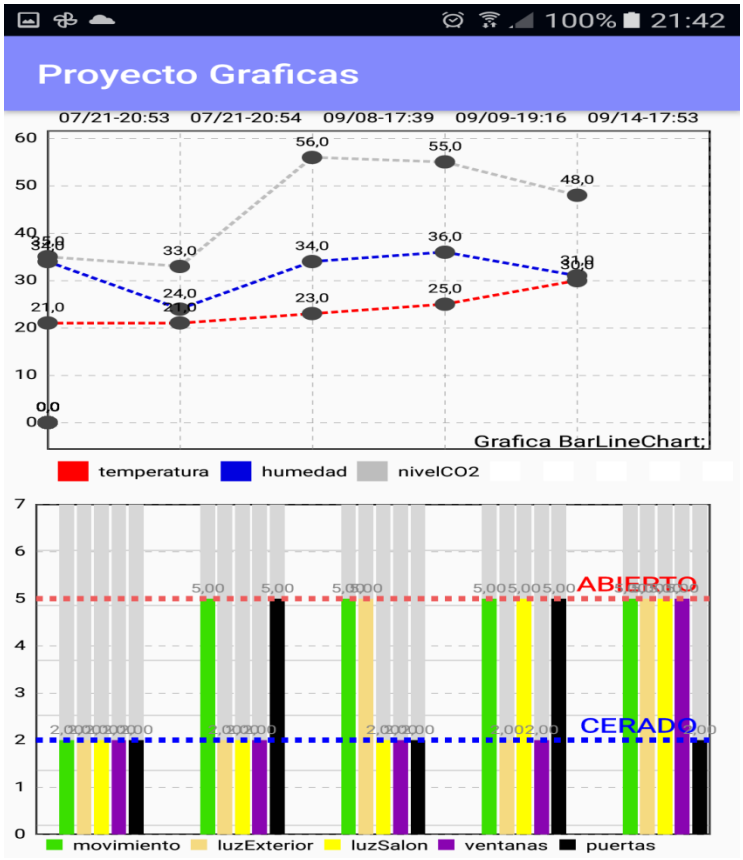
```
            }

            Description description = new Description();

            description.setText("Grafica BarChart; 2.0 representa CERADO
y 5.0 ABIERTO ");

            barchart.setDescription(description);

            barchart.getDescription().setTextSize(12);

            //Marco del BarChart

            barchart.setDrawBorders(true);

            barchart.setBorderWidth(1);

            //Animacion de la presentacion

            barchart.animateXY(2000, 2000);

            barchart.getXAxis().setAxisMinimum(0);

            //para el scroolbar, presenta maximo 3 barras por fecha

            barchart.setVisibleXRangeMaximum(3);

            barchart.invalidate();// refrescar el BarChart

        }catch (Exception bar){
displayExceptionMessage("Exception BarData : "+bar.getMessage());

bar.getMessage(); //Se trata las posibles excepciones

        }}
```

Además de estas funciones, existe una función más necesaria para ambos tipos de graficas getXAxisValues. Esta función es la encargada de extraer y transformar las fechas desde el array de doble dimensión y retornar un array de strings “fechasXAxis”

```
//Valores del Axis X: Las Fechas del servidor

private String[] getXAxisValues(ArrayList<List<String>> datosS) {

    String[] fechasXAxis= new
String[Integer.parseInt(datosS.get(0).get(0))];

    try{

        //Numero de fechas encontradas conforme con la busqueda;
reprezenta tambien el
```

```
        // numero de datos almacenados en cada sublista
        String numDeFechas =datosS.get(0).get(0);
        int numeroDeFechas=Integer.parseInt(numDeFechas);
        String fecha;
        for (int i=0;i<=numeroDeFechas-1;i++){
            //Prueba de funcionalidad
            //displayExceptionMessage("FECHAS FILA 1 :
"+datosS.get(1).get(i));
            //Se añade a la Axis X las fechas quitando el año y los
segundos
           fecha=datosS.get(1).get(i);
            fecha=fecha.substring(5,16);

            //Se remplaza el caracter - por / para una mejor
prezentacion
            fecha=fecha.replace('-','/');
            fecha=fecha.replace(' ','-');
            fechasXAxis[i]=fecha;

            //Prueba de funcionalidad
            //displayExceptionMessage("PRESENTAR FECHA : "+i+"
"+fechasXAxis[i]);
        }
    }catch (Exception e){  displayExceptionMessage("Exception 2 :
"+e.getMessage());
        e.getMessage(); //Se trata las posibles excepciones
    }
    return fechasXAxis;
}
```

5.6.5 Clase BarLineChart:



Esta clase arranca a través del intent de la clase Configuracion y presentara una interfaz combinada entre con las gráficas BarChart y LineChart.

El código es similar a la clase Chart presetada anteriormente, salvo que esta clase no tiene la selección de las gráficas a presenta, las fechas se presentan solamente en la parte superior de la gráfica LineChart, la grafia BarChart no tiene el scroolbar y la gráfica LineChart no tiene el fondo en degradado debajo de cada línea de valores

```
public class BarLineChart extends AppCompatActivity {

    private com.github.mikephil.charting.charts.CombinedChart
barlinechart;

    private LineChart linechart;

    private com.github.mikephil.charting.charts.BarChart barchart;

    public static String servidor;

    //Se crea adicional 2 arrays con los nombres todas las columnas y
con los colores de cada una
```

```
    String
parametros[]={"temperatura","humedad","nivelCO2","movimiento","luzExte
rior","luzSalon","ventanas","puertas"};


    int[] col = {Color.rgb(255, 0, 0), Color.rgb(1, 1, 223),
Color.rgb(189, 189, 189)

            , Color.rgb(58, 223, 0) , Color.rgb(245, 218, 129) ,
Color.rgb(255, 255, 0),

            Color.rgb(137, 4, 177), Color.rgb(1, 1, 1)};

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_bar_line_chart);

        barchart = (com.github.mikephil.charting.charts.BarChart)
findViewById(R.id.barchartx);


        linechart = (LineChart) findViewById(R.id.linechartx);


        //Intent para recuperar los datos


        Intent in = this.getIntent();
 //reprezenta el numero de datos almacenados en cada sublista
  final String numeroDeSublistas = in.getStringExtra("sublistas")

  int contadorSubListas= Integer.parseInt(numeroDeSublistas);

        try {

            //Direccion del Servidor

            servidor = in.getStringExtra("servidor");

            //Prueba de funcionalidad

            //displayExceptionMessage("SERVER RECUPERADO "+servidor);

        }catch (Exception e){

            displayExceptionMessage("Exception obtener server
CLASS:BarLineChart: "+e.getMessage());

            e.getMessage(); //Tratamos las posibles excepciones
```

```
        }

        //Se recupera el array de poziciones

        ArrayList<Integer> pozicion=(ArrayList<Integer>)
getIntent().getSerializableExtra("pozicion");

        //Se crea el ArrayList de doble dimension

        ArrayList<List<String>> datosS = new
ArrayList<List<String>>();

        for (int i = 0; i <= contadorSubListas - 1; i++) {


            //Se crea tantas sublistas, lo que se espera desde la
Activity Configuracion y se guarda cada una en el ArrayList datosS

            String ar = "ar" + i;

            ArrayList lista = (ArrayList<String>)
getIntent().getSerializableExtra(ar);

            datosS.add(lista);

        }


        //Prueba de funcionalidad

        //displayExceptionMessage("Grafica elegida:
"+datosS.get(0).get(1));



        //Para el BarLineChart

xAxisDataBarChart(datosS, pozicion);

        xAxisDataLineChart(datosS, pozicion);

    }

    public void xAxisDataLineChart(ArrayList<List<String>>
datosS,ArrayList<Integer> pozicion ) {

        //BLOQUE ENCARGADO DE CONFIGURAR EL XAXIS E YAXIS

        try{

            final String[] labels = getXAxisValues(datosS);

            XAxis xAxisLine = linechart.getXAxis();
```

```
            //Las lineas verticales del grid de la grafica con las
dimensiones establecidas

            xAxisLine.enableGridDashedLine(10f, 10f, 0f);

            //Dimension orizontal del grid

            xAxisLine.setAxisMaximum(10f);

            xAxisLine.setAxisMinimum(0f);


            //Coloca en el centro de los recuadros las fechas, a
partir de la linea con el valor

            xAxisLine.setCenterAxisLabels(true);

            xAxisLine.setDrawLabels(true);

            xAxisLine.setDrawAxisLine(true);

            //Añade a la axisX el array String de las fechas recibidas
en la funcion getXAxisValues()

            xAxisLine.setValueFormatter(new
IndexAxisValueFormatter(labels));

            YAxis leftAxisLine =linechart.getAxisLeft();


            leftAxisLine.removeAllLimitLines();

   //Las lineas orizontales del grid de la grafica con las dimensiones
establecidas

           leftAxisLine.enableGridDashedLine(10f, 10f, 0f);

            leftAxisLine.setDrawZeroLine(false);

            leftAxisLine.setDrawLimitLinesBehindData(false);

            linechart.getAxisRight().setEnabled(false);

            generateLineData(datosS,pozicion);


        }catch (Exception x){

            displayExceptionMessage("Exception xAxisDataLineChart
CLASS:BarLineChart: "+x.getMessage());

        }

    }
```

```
    public void xAxisDataBarChart(ArrayList<List<String>>
datosS,ArrayList<Integer> pozicion ) {

        //BLOQUE ENCARGADO DE CONFIGURAR EL XAXIS

        try{
          final String[] labels = getXAxisValues(datosS);

            //Prueba de funcionalidad

            //displayExceptionMessage("labels size "+labels.length);

            XAxis xAxisBar = barchart.getXAxis();

            //xAxisBar.setDrawGridLines(false);

            //xAxisBar.setDrawAxisLine(false);

            //xAxisBar.setDrawLabels(false);

            //Coloca en el centro de los recuadros las fechas, a
partir de la linea con el valor

            xAxisBar.setCenterAxisLabels(true);

            xAxisBar.setPosition(XAxis.XAxisPosition.TOP);

            //Añade a la axisX el array String de las fechas recibidas
en la funcion getXAxisValues()

            xAxisBar.setValueFormatter(new
IndexAxisValueFormatter(labels));


            LimitLine ll1 = new LimitLine(5f, "ABIERTO");

            ll1.setLineWidth(4f);

            ll1.enableDashedLine(10f, 10f, 0f);


ll1.setLabelPosition(LimitLine.LimitLabelPosition.RIGHT_TOP);

            ll1.setTextSize(15f);

            ll1.setTextColor(RED);

            LimitLine ll2 = new LimitLine(2f, "CERADO");

            ll2.setLineWidth(4f);

            ll2.enableDashedLine(10f, 10f, 0f);

         ll2.setLabelPosition(LimitLine.LimitLabelPosition.RIGHT_TOP);
```

```
            ll2.setTextSize(15f);

            ll2.setTextColor(BLUE);

            ll2.setLineColor(BLUE);

            YAxis leftAxis = barchart.getAxisLeft();

            leftAxis.removeAllLimitLines();

            leftAxis.addLimitLine(ll1);

            leftAxis.addLimitLine(ll2);

            leftAxis.setAxisMaximum(7f);

            leftAxis.setAxisMinimum(0f);

            leftAxis.enableGridDashedLine(10f, 10f, 0f);

            leftAxis.setDrawZeroLine(false);

            leftAxis.setDrawLimitLinesBehindData(false);

            //Quita los valores presentados en yAxis de la derecha

            YAxis rightAxis = barchart.getAxisRight();

            rightAxis.removeAllLimitLines();

            rightAxis.setDrawLabels(false);


      //Llamo a ejecutar la funcion setDataBarChart()

            generateBarData(datosS,pozicion);

        }catch (Exception x){

            displayExceptionMessage("Exception xAxisDataBarChart
CLASS:BarLineChart: "+x.getMessage());

        }

    }

    private void generateLineData(ArrayList<List<String>> datosS,
ArrayList<Integer> pozicion ) {


        //Se crea el Array para la axix Y

        ArrayList<ILineDataSet> yAxisLineData = new ArrayList();
```

```
        //Se crea la leyenda LineDataSet
        LineDataSet leyendaDataLineBarChart=null;
        try{

            //Se saca el número de sublistas (cuantos checkbox
seleccionados +2 hay) para el bucle
            String numDeListas =datosS.get(0).get(2);
            int numeroDeListas=Integer.parseInt(numDeListas);
            //Se saca el número de filas sleccionadas de la BBDD a
travez de la fecha y hora elegida  para el bucle
            //Reprezenta tambien el numero de datos almacenados en
cada sublista
            String numDeFilas =datosS.get(0).get(0);
            int numeroDeFilas=Integer.parseInt(numDeFilas);
            //Prueba de funcionalidad
            //displayExceptionMessage("LISTAS "+numeroDeListas+"
INDICES SUBLISTAS "+numeroDeFilas);

           //Interesa empezar por la tercera sublista ya que en la
primera (poz.0 del array)

            // se almacena el numero de filas y columnas de la BBDD
            //I en la segunda (poz.1 del array) las fechas
            int contador=0;
            int cont=0;
            for (int i=2;i<=numeroDeListas-1;i++) {
                //El contador representa la pozicion en Axis X
                contador = 0;//Por cada bucle se pone el contador a 0
                //Se crean tantos ArrayList cuatos tipos de datos
seleccionados con los checkbox hay
                ArrayList datosDataLineBarChart = new ArrayList();
                //Bucle para extraer los datos de cada sublista
```

```
                for (int y = 0; y <=numeroDeFilas-1; y++) {

                    //Prueba de funcionalidad

                    //displayExceptionMessage("DATOS FILA "+i+"
POZICION "+y+" : "+datosS.get(i).get(y));

                    String dato = datosS.get(i).get(y);

                    int datoInteger = Integer.parseInt(dato);

                    //Se controla la carga de los datos, en este
apartado solamente para LineChart


                    if(datoInteger>1) {

                        Entry valor = new Entry(contador,
datoInteger);

                        //Se añaden los datos al ArrayList

                        datosDataLineBarChart.add(valor);

                        contador++;

                    }else{

                       Entry valor = new Entry(0, 0);

        //Se añaden los datos al ArrayList
        datosDataLineBarChart.add(valor);    }  }

     //Se controla la presentacion de los colores con su descripcion

if(parametros[pozicion.get(cont)].equals("temperatura")||parametros[po
zicion.get(cont)].equals("humedad")||parametros[pozicion.get(cont)].eq
uals("nivelCO2")) {

                    //Linea de datos

                    //Se añade los datos y la descripcion del color

                    leyendaDataLineBarChart = new
LineDataSet(datosDataLineBarChart, ( parametros[pozicion.get(cont)]));

                    //Color de la linea
  leyendaDataLineBarChart.setColor(col[pozicion.get(cont)]);

                }else{

                    //Se añade los datos y la descripcion del color
```

```
                    leyendaDataLineBarChart = new
LineDataSet(datosDataLineBarChart, "");


                    leyendaDataLineBarChart.setColor(Color.WHITE);

                }

                //Linea de datos dibujo

                leyendaDataLineBarChart.enableDashedLine(10f, 5f, 2f);

                //Linea de datos grueso

                leyendaDataLineBarChart.setLineWidth(2f);

                //Puntos de las lineas

                leyendaDataLineBarChart.setCircleColor(Color.DKGRAY);

                leyendaDataLineBarChart.setCircleRadius(5f);

                //Los puntos estan rellenos

                leyendaDataLineBarChart.setDrawCircleHole(false);

                //Tamaño de las letras de la leyenda

                leyendaDataLineBarChart.setValueTextSize(9f);


          //Habilita el fondo de color por debajo de las lineas
        //leyendaDataLineBarChart.setDrawFilled(true);

                leyendaDataLineBarChart.setFormLineWidth(2f);

                leyendaDataLineBarChart.setFormLineDashEffect(new
DashPathEffect(new float[]{10f, 5f}, 2f));

                //Tamaño del recuadro de color da la leyenda

                leyendaDataLineBarChart.setFormSize(15.f);

                yAxisLineData.add(leyendaDataLineBarChart);

                cont++;

            }

        }catch (Exception e){

            displayExceptionMessage("Exception LineData
CLASS:BARLINECHART : "+e.getMessage());
```

```
            e.getMessage(); //Trato las posibles excepciones
        }
        LineData datosLDataLineBarChart = new LineData(yAxisLineData);
        try{
            linechart.animateXY(4000, 4000);
            //Se dibuja el mChart
            linechart.setData(datosLDataLineBarChart);
            Description description = new Description();
            description.setText("Grafica BarLineChart;");
            linechart.setDescription(description);
            linechart.getDescription().setTextSize(12);
            //Marco del LineChart
            linechart.setDrawBorders(true);
            linechart.setBorderWidth(1);
            // PRUEBA DESPLAZAR GRAFICA
            linechart.setScaleMinima((float)
datosLDataLineBarChart.getXMin() / 7f, 1f);

            linechart.moveViewTo(0, 7, YAxis.AxisDependency.LEFT);
            //Dimension minimo y maximo, dibuja tantas lineas
verticales cuantas fechas hay disponibles
            linechart.getXAxis().setAxisMinimum(0);
linechart.getXAxis().setAxisMaximum(Integer.parseInt(datosS.get(0).get
(0)));
            //Coloca las fechas desde la primera linea vertical del
grid
linechart.getXAxis().setLabelCount(Integer.parseInt(datosS.get(0).get(
0))+1, true);
            linechart.invalidate();// refrescar el linechart
        }catch (Exception e){
```

```
            displayExceptionMessage("Excepcion linechart
CLASS:BARLINECHART : "+e.getMessage());


        }

    }

    private void generateBarData(ArrayList<List<String>> datosS,
ArrayList<Integer> pozicion ) {

        //Se crea el Array para la axix Y

        ArrayList<IBarDataSet> yAxix = new ArrayList();

        //Se crea la leyenda BarDataSet

        BarDataSet leyenda=null;

        //ESTE BLOQUE SE ENCARGA DE EXTRAER LA INFORMACION DEL
ARRAYLIST Y COLOCARLA EN EL BARDATASET (VALORES YAXIS)

        try{

            //Se saca el numero de sublistas (cuantos checkbox
seleccionados +2 hay) para el bucle

            String numDeListas =datosS.get(0).get(2);

            int numeroDeListas=Integer.parseInt(numDeListas);


            //Se saca el numero de filas sleccionadas de la BBDD a
travez de la fecha y hora elegida  para el bucle


            //Reprezenta tambien el numero de datos almacenados en
cada sublista

            String numDeFilas =datosS.get(0).get(0);

            int numeroDeFilas=Integer.parseInt(numDeFilas);

            //Prueba de funcionalidad

            //displayExceptionMessage("LISTAS "+numeroDeListas+"
INDICES SUBLISTAS (numeroDeFilas) "+numeroDeFilas);

            //Interesa empezar por la tercera sublista ya que en la
primera (poz.0 del array)

            // se almacena el numero de filas y columnas de la BBDD
```

```
            //I en la segunda (poz.1 del array) las fechas
            int contador=0;
            int cont=0;
            for (int i=2;i<=numeroDeListas-1;i++) {
                //El contador representa la pozicion en Axis X
                contador = 0;//Por cada bucle se pone el contador a 0
                //Se crean tantos ArrayList cuatos tipos de datos
seleccionados con los checkbox hay
                ArrayList datos = new ArrayList();
                try{
                    //Bucle para extraer los datos de cada sublista
                    for (int y = 0; y <= numeroDeFilas - 1; y++) {
                        //Prueba de funcionalidad
                        //displayExceptionMessage("DATOS FILA "+i+"
POZICION "+y+" : "+datosS.get(i).get(y));
                        String dato = datosS.get(i).get(y);
                        int datoInteger = Integer.parseInt(dato);
        // Se cambia el valor para una mejor presentacion
                        if (datoInteger == 0) {
                           datoInteger = 2;  }
      // Se cambia el valor para una mejor presentación
                        if (datoInteger == 1) {
                            datoInteger = 5;
                        }

     //Se controla la carga de los datos, en este apartado solamente
para BarChart
                        if(datoInteger<6) {
            BarEntry valor = new BarEntry(contador, datoInteger);
```

```
                //Se añaden los datos al ArrayList
                            datos.add(va1or);
                            contador++;
                        }   }
                }catch(Exception ex){
                    displayExceptionMessage("Exception bucle for :
"+ex.getMessage());
                    ex.getMessage();
//Tratamos las posibles excepciones
  }
                //Se añade por cada linea la descripccion de la
columna y el color corespondiente
                leyenda = new
BarDataSet(datos,parametros[pozicion.get(cont)]);
                leyenda.setColor(col[pozicion.get(cont)]);
                //Tamaño de las letras de la leyenda
                leyenda.setValueTextSize(9f);
                //Quita los valores de los puntos de cada columna
                //leyenda.setDrawValues(false);


//Color del texto de los valores de los puntos de cada columna

                leyenda.setValueTextColor(GRAY);
                yAxix.add(leyenda);
                cont++;
            }
            //Prueba de funcionalidad
            //displayExceptionMessage("CONTADOR "+contador+" CONT
"+cont);
```

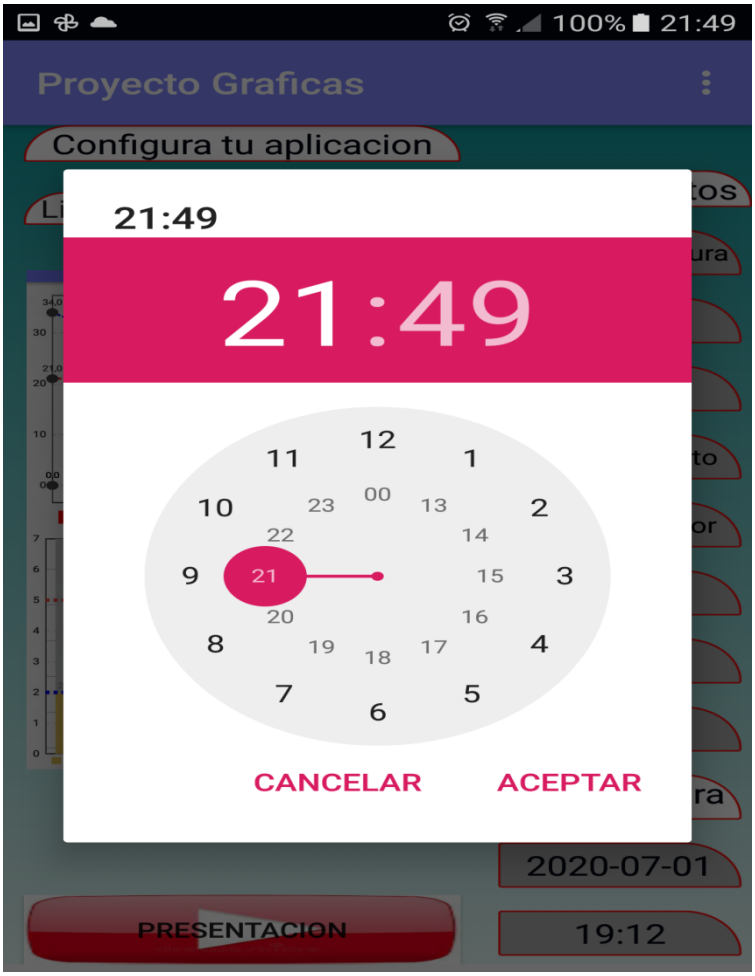
```
        }catch (Exception e){
            displayExceptionMessage("Exception generateBarData
CLASS:BARLINECHART : "+e.getMessage());
            e.getMessage(); //Tratamos las posibles excepciones
        }

        //BLOQUE ENCARGADO DE CONFIGURAR EL BARCHART PARA DEPUES
PASARLO AL XML
        try {
            BarData dataBarChart = new BarData(yAxix);
         float groupSpace = 0.06f;//espacio entre los grupos de barras
         float barSpace = 0.08f; // espaccio entre baras
          float barWidth = 0.48f; // Grueso de las baras
       //Configura el BarData con el grueso de las barras establecido
            dataBarChart.setBarWidth(barWidth);
            barchart.setData(dataBarChart);
 //Se controla la gráfica en función del número de datos seleccionados
            if(Integer.parseInt(datosS.get(0).get(2))==3){
                barWidth = 0.25f; // Grueso de las baras
                dataBarChart.setBarWidth(barWidth);

//  Integer.parseInt(datosS.get(0).get(0))  es  el  número  de  entradas
//Presenta el número de fechas encontradas colocándolas en el TOP

barchart.getXAxis().setAxisMaximum(Integer.parseInt(datosS.get(0).get(
0)));
            }else {
                barchart.groupBars(0, groupSpace, barSpace);
   // Integer.parseInt(datosS.get(0).get(0)) es el número de entradas
 //Presenta el número de fechas encontradas colocándolas en el TOP
```

```
barchart.getXAxis().setAxisMaximum(barchart.getBarData().getGroupWidth
(groupSpace, barSpace) * Integer.parseInt(datosS.get(0).get(0)));

            }

             Description description = new Description();

             description.setText("Grafica LineBarChart; 2:CERADO
5:ABIERTO ")

             barchart.setDescription(description);

             //Se quita la descripción

             barchart.getDescription().setEnabled(false);

             //Dibuja las columnas del grid

barchart.getXAxis().setLabelCount(Integer.parseInt(datosS.get(0).get(0
))+1, true);

             //Marco del BarChart

             barchart.setDrawBorders(true);

             barchart.setBorderWidth(1);

             barchart.setDrawBarShadow(true);

             barchart.getXAxis().setAxisMinimum(1);

             barchart.getXAxis().setDrawGridLines(false);

             barchart.getXAxis().setDrawAxisLine(false);

             //Se quita la presentación de las fechas


             barchart.getXAxis().setDrawLabels(false);

             //Animacion de la presentacion

             barchart.animateXY(2000, 2000);

             //para el scroolbar, presenta maximo 4 barras por fecha

             //barchart.setVisibleXRangeMaximum(4);

             barchart.invalidate();// refrescar el BarChart

         }catch (Exception bar){

             displayExceptionMessage("Exception BarData
CLASS:BARLINECHART : "+bar.getMessage());
```

```
            bar.getMessage(); //Se trata las posibles excepciones
        }
    }
    //Valores del Axis X: Las Fechas del servidor
    private String[] getXAxisValues(ArrayList<List<String>> datosS) {
        String[] fechasXAxis= new
String[Integer.parseInt(datosS.get(0).get(0))];
        try{
            //Numero de fechas encontradas conforme con la busqueda;
reprezenta tambien el
            // número de datos almacenados en cada sublista
            String numDeFechas =datosS.get(0).get(0);
            int numeroDeFechas=Integer.parseInt(numDeFechas);
            String fecha;
            for (int i=0;i<=numeroDeFechas-1;i++){
                //Prueba de funcionalidad
                //displayExceptionMessage("FECHAS FILA 1 :
"+datosS.get(1).get(i));
     //Se añade a la Axis X las fechas quitando el año y los segundos
                fecha=datosS.get(1).get(i);
                fecha=fecha.substring(5,16);
      //Se remplaza el carácter - por / para una mejor presentación
                fecha=fecha.replace('-','/');
                fecha=fecha.replace(' ','-');
                fechasXAxis[i]=fecha;
                //Prueba de funcionalidad
                //displayExceptionMessage("PRESENTAR FECHA : "+i+"
"+fechasXAxis[i]);
            }
```

```
        }catch (Exception e){
            displayExceptionMessage("Exception 2 : "+e.getMessage());
            e.getMessage(); //Se trata las posibles excepciones
        }
        return fechasXAxis;
    }
    //Mensajes adicionales
    public void displayExceptionMessage(String msg)
    {  Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
    }
}
```

5.6.6 Clase TimePickerFragment:



Esta clase, extendida a DialogFragment, es la encargada de presentar el reloj con la hora actual como dialogo con el usuario y ofrece la posibilidad de elegir la hora y el minuto deseado para la selección de los datos del servidor.

Además de su extensión al DialogFragment que permite la elección de la hora y minuto, implementa TimePickerFragment.OnTimeSetListener necesario la escucha cuando el usuario selecciona la hora y el minuto.

```
public class TimePickerFragment extends DialogFragment

        // El propio framento es un listener así que se

        // debe implementar: OnTimeSetListener

        implements TimePickerDialog.OnTimeSetListener {

    // EditText al que se le asocia este Picker
    private EditText editText;
```

Es necesario pasarle el EditText al constructor de la clase pero no se puede modificar el constructor de un fragmento, así que se debe crear un método estático que devuelva una instancia de esta clase, pasándole a este "constructor estático" el EditText al que está asociado y en el que se necesita que se escriban la hora y minuto al ser seleccionadas .

```
    public static TimePickerFragment newInstance(EditText editText) {

        TimePickerFragment fragment = new TimePickerFragment();

        fragment.setEditText(editText);

        return fragment;

    }
```

Se debe crear un setter ya que el "constructor" es estático

```
    public void setEditText(EditText editText){

        this.editText = editText;

    }

    @NonNull

    @Override

    public Dialog onCreateDialog(Bundle savedInstanceState) {
```

```
        final Calendar c = Calendar.getInstance();

        int hour = c.get(Calendar.HOUR_OF_DAY);

        int minute = c.get(Calendar.MINUTE);
        int seconds=c.get(Calendar.SECOND);


          return new TimePickerDialog(getActivity(), this, hour,
minute,

                  DateFormat.is24HourFormat(getActivity()));

    }
```

Cuando se seleccione una hora, el evento llamará a este método donde se da el formato de la hora y minute elegido

```
    @Override

    public void onTimeSet(TimePicker view, int hourOfDay, int minute)
{

        String selectedHour = String.format("%02d", hourOfDay) + ":" +
String.format("%02d", minute) ;
this.editText.setText(selectedHour);  }
}
```

5.6.7 Clase DatePickerDialog:

Esta clase, extendida a DialogFragment, es la encargada de presentar el calendario actual como dialogo con el usuario y ofrece la posibilidad de elegir la fecha deseada para la selección de los datos del servidor.

Además de su extensión al DialogFragment que permite la elección de la fecha, implementa DatePickerDialog.OnDateSetListener necesario la escucha cuando el usuario selecciona la fecha.

```
public class DatePickerDialog extends DialogFragment

        // El propio framento es un listener así que

        // debe implementar: OnDateSetListener

        implements android.app.DatePickerDialog.OnDateSetListener {

    // EditText al que se le asocia este Picker

    private EditText editText;
```

Es necesario pasarle el EditText al constructor de la clase pero no se puede modificar el constructor de un fragmento, así que se debe crear un método estático que devuelva una instancia de esta clase, pasándole a este "constructor estático" el EditText al que está asociado y en el que se necesita que se escriban las fechas al ser seleccionadas.

```
 public static DatePickerDialog newInstance(EditText editText) {

        DatePickerDialog fragment = new DatePickerDialog();

        fragment.setEditText(editText);

        return fragment;

    }
```

Se debe crear un setter ya que el "constructor" es estático

```
    public void setEditText(EditText editText){

        this.editText = editText;

    }

    @Override

    public Dialog onCreateDialog(Bundle savedInstanceState) {
```

```
        // Ponemos la fecha actual para el datepicker
        Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);
        //Si ya hemos seleccionado una fecha en el picker y sigue
        //el formato que le hemos puesto nosotros (en este caso
        //dd/MM/yyyy) muestra la fecha seleccionada en el picker en
        //lugar de la fecha actual
        if(this.editText.getText().toString().length()>0) {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-
dd");
            Date parsedDate = null;
            try {
                parsedDate =
formatter.parse(this.editText.getText().toString());
                c.setTime(parsedDate);
                year = c.get(Calendar.YEAR);
                month = c.get(Calendar.MONTH);
                day = c.get(Calendar.DAY_OF_MONTH);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }
        // Devolver una instancia del DatePickerDialog
        // indicando quién es el listener del picker (this)
        // y la fecha pre-seleccionada (year, month, day)
        return new android.app.DatePickerDialog(getActivity(), this,
year, month, day);
```

```
    }

    //Cuando se seleccione una fecha, el

    // evento llamará a este método


    public void onDateSet(DatePicker view, int year, int month, int
day) {

        String selectedDate = String.format("%04d", year) + "-" +
String.format("%02d", (month+1)) + "-" + String.format("%02d", day);

        this.editText.setText(selectedDate);

    }
}
```

5.6.8 Clase MyReciver:

Esta clase no tiene interfaz. Es extendida a BroadcastReceiver, lo que supone que es un escuchador de broadcast, enviado desde la clase Configuracion. Se recibe en el método onReceive.

```
public class MyReceiver extends android.content.BroadcastReceiver {

     static String campo;
```

Se recuperan las variables ruta y tipo desde la clase Configuración, dado que son públicas.

Se prefiere de esta forma, pero es posible recuperarlas, forzando un intent a traves de onReceive

```
    final String ruta=Configuracion.ru;
    final String tipo=Configuracion.tipo;

    @Override
    public void onReceive(Context context, Intent intent) {
```

La variable campo se cambia de valor desde la clase AuxAlarma, una vez realizado el Intent service para arrancar la clase AuxAlarma. Esto implica que en la primera llamada a esta clase la variable “campo” tendrá el valor null.

```
        try {
            if (campo!=null&&campo.equals("1")) {
```

Se llama a la clase MyAlertDialog para ejecutar la alerta ya que es imprescindible una clase que extiende a Activity para los diálogos.

```
Intent i=new Intent(context.getApplicationContext(),MyAlertDialog.
class);
```

Se crea una pila de actividades a través de Intent para la clase MyAlertDialog

```
i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

Se arranca la clase MyAlertDialog

```
                context.startActivity(i);
            }
```

Se crea un nuevo intent para arancar el IntentService AuxAlarm que se ejecutara en segundo plano

```
            Intent service = new Intent(context,  AuxAlarma.class);
```

Se envía los campos tipo y ruta, necesarios para la conexión al servidor y recuperación del string que interesa

```
            service.putExtra("movim", tipo);
            service.putExtra("ruta", ruta);
```

Se arranca el servicio

```
            context.startService(service);

  //Prueba de funcionalidad
  //Toast.makeText(context," CAMPO en MyReceiver "+campo ,
Toast.LENGTH_LONG).show();
```

Después se para el servicio ya que la peticion al servidor esta en ejecución

```
            context.stopService(service);

        }catch (Exception e){
            Toast.makeText(context, "Exception en MyReceiver
"+e.getMessage(), Toast.LENGTH_LONG).show();
        }


        //Prueba de funcionalidad
        //Toast.makeText(context,"CAMPO "+campo,
```

```
Toast.LENGTH_LONG).show();
    }
}
```

5.6.9 Clase AuxAlarma:

Esta clase no tiene interfaz. Es extendida a IntentService, lo que implica que es un servicio que se ejecuta en segundo plano. Se prefiere un servicio en segundo plano ya que de esta forma no se consumen los recursos necesarios para la ejecución del proceso de la aplicación, si no se ejecutará en cuanto los recursos se liberan.

Los métodos sobrescritos:

`onCreate(),onStartCommand (Intent intent, int flags, int startId), onBind(Intent intent), onHandleIntent( Intent intent)` y `onDestroy()`

Son creados por defecto al crear esta clase.

En el método onStartCommand se reciben los Strings
"`campoParaRecuperar`" y "`ruta`"
A través del Intent del mismo método, utilizando como claves las palabras "**`movim`**" y "**`ruta`**" por igual que en la clase MyReceiver.

Se crea la cola del volley para la comunicacion con el servidor
`final RequestQueue cola = Volley.newRequestQueue(AuxAlarma.this);`

Se llama a la ejecución la función "recuperarInfo" pasándole los 2 Strings y la cola del vóley.

Por lo último con "return ***`START_STICKY`***" se indica al sistema que reinicie el servicio si ha sido parado por algun problema de recursos.

En el método "recuperarInfo" se crea la variable "respuesta" que se esperara desde el constructor de la clase Aux, y al igual que en las clases Login y Configuracion, se trata de recuperar un String en este caso "campo" a través del JSONObject, pasándole como clave la variable "`recuperar`" . De esta manera se va a buscar el dato exacto que interesa entre los datos recibidos desde el servidor.

Al final del bloque try/catch se le pasa al constructor de la clase Aux la variable ruta que es la ruta al fichero consulta.php y se espera recuperar la respuesta desde el servidor.

```
//Llamo al constructor de la clase Auxiliar
r = new Auxiliar(ruta,respuesta);
```

En este caso antes de añadir a la cola del vóley una nueva petición, se prefiere cancelar cualquier respuesta pendiente.

```
cola.cancelAll(respuesta);
```

Por lo último, en el método procesarResultados, se cambia el valor del la variable static "*campoA*" al valor recibido desde JSONObject (variable "campo" de la función onResponse) para poder acder y cambiar el valor de la variable "*campo*" de la clase MyReceiver

```
public class AuxAlarma extends IntentService{
    private Auxiliar r;
    static String campoA="2";

    /**
     *
     * @param name Used to name the worker thread, important only for
debugging.
     */
    public AuxAlarma(String name) {
        super(name);
    }

    public AuxAlarma() {
        super("AuxAlarma");
    }


    @Override
    public void onCreate() {
        //Prueba de funcionalidad
        //displayExceptionMessage("Servicio creado" );
    }

    @Override
    public int onStartCommand (Intent intent, int flags, int startId)
{


        //Se reciben las variables necesarias desde la clase
MyReceiver a traves del intent
        final String campoParaRecuperar =
intent.getStringExtra("movim");
        final String ruta= intent.getStringExtra("ruta");

        //Se crea la cola del volley (comunicacion con el servidor)
        final RequestQueue cola =
Volley.newRequestQueue(AuxAlarma.this);

        //Prueba de funcionalidad
        //displayExceptionMessage("Empieza onStartCommand CAMPO
"+campoParaRecuperar );
```

```
        //Se pasa las variables a la funcion recuperarInfo()
        recuperarInfo(ruta,campoParaRecuperar,cola);
        // Se reinicia el servicio si ha sido parado por el sistema
por algun problema de recursos
        return START_STICKY;
    }


    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    protected void onHandleIntent( Intent intent) { }

    @Override
    public void onDestroy() {
        //Prueba de funcionalidad
         //displayExceptionMessage("Servicio parado"  );
    }


    /**
     * @param ruta
     * @param recuperar
     * @param cola
     */
    public void recuperarInfo(final String ruta, final String
recuperar,final RequestQueue cola){
        Response.Listener<String> respuesta = new
Response.Listener<String>() {

            @Override
            public void onResponse(String response) {

                try{

                    //Creo el objeto jsonRespuesta para la respuesta
del servidor
                    JSONObject jsonRespuesta = new
JSONObject(response);
                    String campo=jsonRespuesta.getString(recuperar);
                    //Pasar el resultado a la funcion
                    procesarResultados(campo);

                }catch (JSONException e){
                    displayExceptionMessage("JSONException "
+e.getMessage() );
                    e.getMessage(); //Trato las posibles excepciones

                }
            }

        };

        //Llamo al constructor de la clase Auxiliar
        r = new Auxiliar(ruta,respuesta);
        // Utilizo Volley para la comunicacion. En este caso cola se
```

```
vasea antes de añadir la nueva respuesta
        cola.cancelAll(respuesta);
        cola.add(r);

    }

    /**
     * @param campoRecuperado
     * Recibe el string recuperado del servidor y cambia el valor en
la clase MyReceiver
     */
    public void procesarResultados(String campoRecuperado ){

        campoA = campoRecuperado;
        //Se cambia el valor del String campo de la clase MyReceiver
con cada iteracion
        MyReceiver.campo = campoA;

        //Prueba de funcionalidad
        // displayExceptionMessage("Campo A "+campoA);

    }

    /**
     * @param msg
     * Mensages adicionales; se utiliza en el desarollo del codigo
para encontrar los errores
     */
    public void displayExceptionMessage(String msg)
    {
        Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
    }
}
```

5.6.10 Clase Auxiliar:

Esta clase no tiene interfaz. Es extendida a StringRequest y se utiliza para la comunicación con el servidor SQL a través de los ficheros php del servidor Apache.

Utiliza un Map de strings "**parametros**" que se define como private, que envía y recibe la infermocion desde y hacea el servidor Apache

Se utilizan 3 constructores para la comunicacion con las clases AuxAlarma, Login y Configuracion. Cada constructor utiliza el método "super" que hace referencia a la superclase "StringRequest". A su vez este método utiliza el "`Request.Method.POST` " para enviar los datos recibidos por el constructor. Para el envio se necesita además la ruta donde serán enviados estos datos, un listener y el error que podría proporcionar la conexión

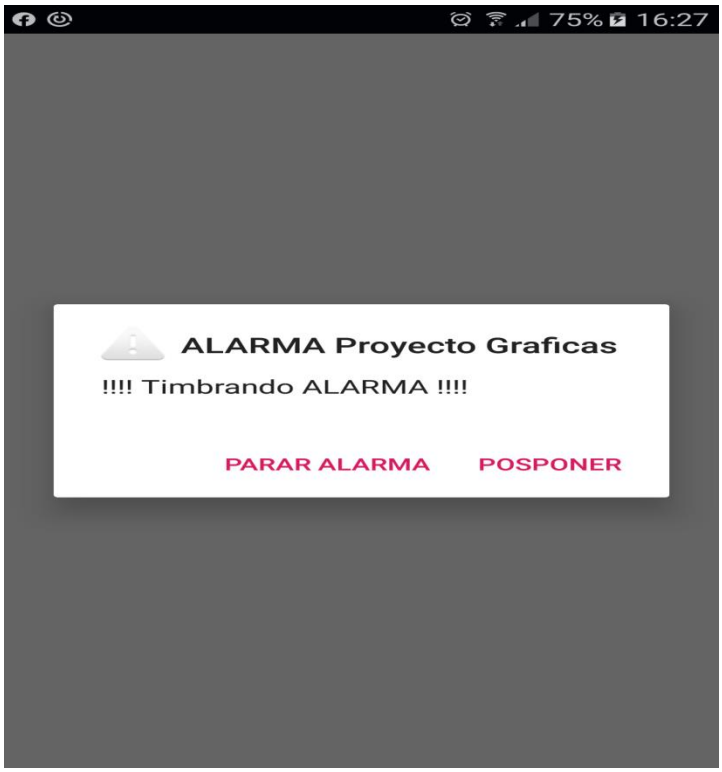
Esta ruta es recibida a través de la variable "`server`" siendo la ruta a los ficheros php que se necesita por cada clase, es decir cambia dependiendo de la clase que accede al constructor adecuado.
La variable "listener" es la "respuesta" esperada en cada clase. Ademas se puede observar que el error "errorListener" se pone a null e 2 de los constructores, ya que en sus casos no será necesario tratar este error de conexión.

Por lo último tenemos el método sobrescrito getParm que retorna el Map de parámetros.

```
public class Auxiliar extends StringRequest{
    String usuario="";//Se ponen a nada por si se repite el Login. De
esta forma no guarda el valor antiguo
    String clave="";//Se ponen a nada por si se repite el Login. De
esta forma no guarda el valor antiguo


    private Map<String, String> parametros; //Para el envio de datos

    /**
     * @param ruta
     * @param listener
     * Constroctor para el servicio AuxAlarma
     */
        public Auxiliar(String ruta, Response.Listener<String>
listener) {
            super(Request.Method.POST, ruta, listener,  null);
            parametros = new HashMap<>();
        }

    /**
     * @param usuario
     * @param clave
     * @param server
     * @param listener
     * @param errorListener
     * Constructor para la clase Login
     */
        public Auxiliar( String usuario, String clave, String server,
Response.Listener<String> listener, Response.ErrorListener
errorListener) {
            super(Request.Method.POST, server, listener,
errorListener);
            this.usuario=usuario;
            this.clave=clave;
            parametros = new HashMap<>();
            parametros.put("usuario",usuario+"");
            parametros.put("clave",clave+"");
        }

    /**
     * @param fechahora
     * @param server
     * @param listener
     * Constructor para la clase Configuracion
     */
        public Auxiliar( String fechahora, String server,
Response.Listener<String> listener) {
            super(Request.Method.POST, server, listener, null);
            parametros = new HashMap<>();
            parametros.put("fechahora",fechahora+"");
        }
```

```
    // Sobescribir el metodo getParams()
        @Override
        protected Map<String, String> getParams() {
            return parametros;
        }

}
```

5.6.11 Clase MyAlertDialog:



Esta clase esta extendida a Activity. A pesar de que tiene interfaz, (el archivo activity_my_alert_dialog.xml), no interesa presentar su interfaz si no solamente el dialogo presentado cuando se activa la alarma, además del timbre "alert.mp3" del dirctorio raw. Por eso en el método onCreate se coloca la siguente línea: requestWindowFeature(Window.***FEATURE_NO_TITLE***); que indicara al sistema que esconda la interfaz entera

Se utiliza un MediaPlayer para acceder al archivo de sonido y se arranca con start()

Se crea un dialogo "AlertDialog.Builder" para presentar el informe y para que el usuario pueda elegir entre parar o posponer la alarma

Este dialogo tiene:

-El mensaje

-El titulo

-Un icono

-El botón Negative (Parar Alarma) que a través de su método sobescrito onClick al pulsarlo para el MediaPlayer, cancela el PedingIntent de la clase Configiguracion, para el servicio AuxAlarma y cierra esta clase

-El botòn Positive (Posponer) que a través de su método sobescrito onClick al pulsarlo para el MediaPlayer y ciera esta clase, pero se seguirán ejecutando los PedingIntent de la clase Configuracion y el servicio de la clase AuxAlarma.

Por lo ultimo se crea y se presenta el dialogo.

Esta clase se ejecutara cada vez que es activada desde MyReceiver si se cumple la condición de que el campo cambia al valor “1”

```
public class MyAlertDialog extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE); //Esconde el
titulo y la interfaz entera
        setContentView(R.layout.activity_my_alert_dialog);

final MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.alert);
        mediaPlayer.start();
         //Configuracion de la alerta
        AlertDialog.Builder Builder=new AlertDialog.Builder(this)
                .setMessage("!!!! Timbrando ALARMA !!!!")
                .setTitle("ALARMA Proyecto Graficas")
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setNegativeButton("PARAR Alarma", new
DialogInterface.OnClickListener() {
                    @Override
            public void onClick(DialogInterface dialog, int which) {
                        //Parar la reproduccion del soñido
                        mediaPlayer.stop();
                        Configuracion c=new Configuracion();
                        c.apagarAlarma();
                        //Se cancela el PendingIntent

Configuracion.manager.cancel(Configuracion.pIntent);
                        //Se para el servicio del segundo plano
       stopService(new Intent(MyAlertDialog.this, AuxAlarma.class));

                        //Se ciera esta clase
                        MyAlertDialog.this.finish();

                    }
                })
                .setPositiveButton("POSPONER", new
DialogInterface.OnClickListener() {
                    @Override
            public void onClick(DialogInterface dialog, int which) {
                        mediaPlayer.stop();
                        MyAlertDialog.this.finish();
```

```
                    }
                });
        AlertDialog alertDialog=Builder.create();
        alertDialog.show();

    }
}
```

6. Justificación de la propuesta técnica

Actualmente existen varias aplicaciones capaces de presentar el valor de un sensor, a través de Android, pero no existen muchas que presenten el historial de los datos de estos sensores, y permitan el acceso a esa información de forma sencilla y más aún en formato de gráficas, y por estas razones, he pensado que una aplicación así puede resultar muy útil si lo que se desea es acceder al historial de esa información o trabajar con ella, tal y como se expuso más arriba en el punto 3 (Estudio de necesidades de la empresa) y además disponer de esa información en cualquier momento y lugar, desde nuestro teléfono móvil, así como también disponer de una alerta por si queremos conocer exactamente si se produce un cambio.

7. Implantación

El trabajo es práctico, y consta en desarrollar el código, así como también la instalación de los servidores, creación de la base de datos y reconfiguración del firewall del PC. Salvo la parte de la instalación física de los sensores que por razones de costes y practicidad se han simulado.

I. Instalación de un servidor web, junto con un servidor MySql y la base de datos donde se va a almacenar los datos necesarios. En mi caso los servidores los instalo en mi PC.
II. Configuración de las reglas necesarias en el firewall del PC, para la comunicación (reglas de entrada, salida, puertos etc.)
III. Ficheros .php capaces de tratar la información, recibida tanto desde la aplicación como desde el servidor MySql y pasarla al correspondiente nivel.
IV. Elaborar dos ficheros .php que simulan la entrada de datos desde los sensores a la base de datos.
V. Programación de la aplicación en Android Studio con sus Activities y clases necesarias
VI. Realizar diferentes pruebas para verificar su funcionalidad.

8. Conclusiones

En primer lugar quisiera resaltar que al desarrollar este proyecto he podido trabajar la programación y profundizar en otras ramas de la informática como son las redes, Sistemas.

Por otra parte he reflexionado sobre la importancia de la información en el ámbito de las empresas y su tratamiento, desde su accesibilidad hasta su análisis y obtención de resultados (aprovechamiento), y de cómo las tecnologías de la información han transformado la forma y el funcionamiento de las empresas, la forma en que estas aprovechan recursos que antes pasaban desapercibidos y que les permiten mejorar en muchos aspectos.

Constatar que la programación en conjunto con otras ramas de la informática puede aportar y aporta mucho a los procesos operativos de las empresas en muchos de sus ámbitos como puede ser el almacenamiento y análisis de su propia información, la seguridad física de sus instalaciones, siendo casi indiferente la actividad a la cual se dedique, para cada tipo de procesos productivos se puede dar una respuesta tecnológica que le puede ser de gran utilidad.

También he tenido la oportunidad de aprender nuevas como por ejemplo la comunicación entre Android con el servidor Apache por medio de los ficheros php, la clase AlarmManager del sistema para la configuración de las alarmas y la creación de diferentes tipos de gráficas en Android.

9. Propuestas de mejora

- Si la aplicación tuviese un uso real, se podría configurar la aplicación te tal manera que el usuario pueda cambiar los nombres de las variables (sensores) e incluso añadir o quitar sensores de la base de datos

- Que el usuario pueda acceder a la base de daatos y poder cambiar su contraseña. Ademas la contraseña se envia en texto plano, lo que supone un riesgo de seguridad, la contraseña se debería enviar cifrada y descifrarla en el servidor

- Tambien seria posible configurar los servidores y la base de datos en la nube, sin tener la necesidad de tener instalado en la casa en nuestro PC.

10. Fuentes

https://www.anychart.com/es/technical-integrations/samples/android-charts/

https://stackoverflow.com/questions/3918517/calling-startactivity-from-outside-of-an-activity-context

https://android.developpreference.com/article/16698438/Activity+as+dialog+from+BroadcastReceiver+over+another+activity

https://sites.google.com/view/kkpqgtkuqp/com-github-mikephil-charting-charts-barchart-example

https://www.programcreek.com/java-api-examples/?class=com.github.mikephil.charting.charts.LineChart&method=setPinchZoom

https://academiaandroid.com/clase-alarmmanager-planificacion-tareas-android/

https://sodocumentation.net/es/android/topic/2800/voleo

https://academiaandroid.com/videotutorial-proyecto-app-android-con-conexion-remota-a-mysql/

https://www.youtube.com/watch?v=IIPz9JuPrcI

https://developer.android.com/guide/background?hl=es-419

http://www.androidcurso.com/index.php/recursos/38-unidad-8-servicios-notificaciones-y-receptores-de-anuncios/288-un-servicio-para-ejecucion-en-segundo-plano