

INSTITUTO TECNOLÓGICO DE CHILPANCINGO

Ingeniería en Sistemas Computacionales

**Proyecto Scanner de Wifi**

Taller de Cómputo Móvil

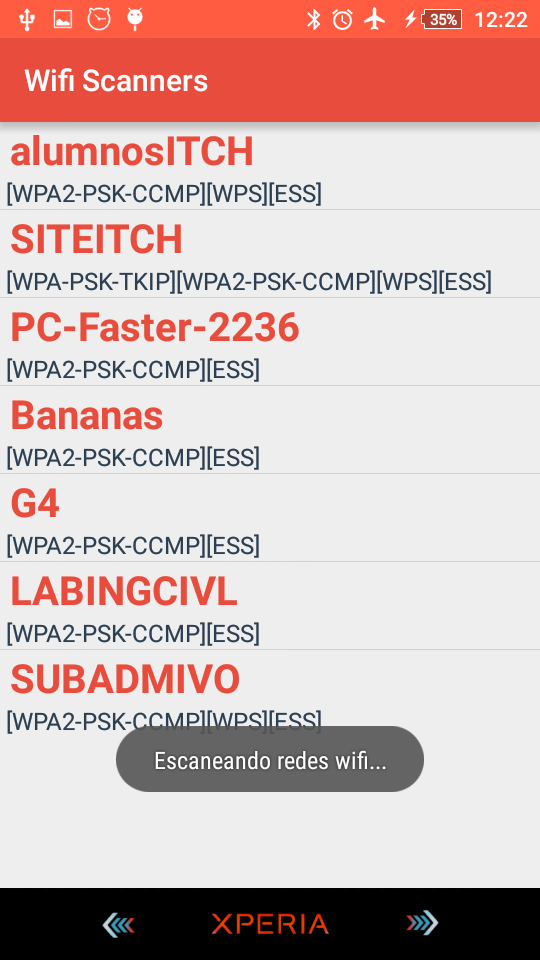
M. C. José Mario Martínez Castro

Alumno: Fernando Lucena Calixto

**Chilpancingo, Gro. Julio del 2016**

Proyecto Scanner de Wifi

Propondremos la realización de una aplicación que haga una búsqueda de las redes Wifi que tenemos a nuestro alrededor y que nos muestre sus parámetros, como por ejemplo el ssid y qué seguridad utilizan. La aplicación resultante debe tener la siguiente apariencia.

[](https://github.com/fercho0/Wifi_Scanners_Netword/blob/master/img/w1.png) [](https://github.com/fercho0/Wifi_Scanners_Netword/blob/master/img/w2.png)

Nos ponemos manos a la obra y creamos un proyecto en Android cuyo nombre sea [Wifi\_Scanners\_Neteork](https://github.com/fercho0/Wifi_Scanners_Netword/blob/master), como paquete package com.example.forevertucode.wifi\_scanners\_netword, su target Android 5.0 y no nos olvidamos de indicarle que queremos una clase principal llamada MainActivity.java.

## **Diseñando la interfaz de la actividad activity\_main.xml**

Una vez tenemos nuestro proyecto creado nos vamos a poner a modelar la interfaz de la aplicación. Vamos a empezar por la primera actividad (clase MainActivity) que, tal y como observamos en la imagen anterior, tendrá como elementos una imagen (ImageView) y un botón con funcionalidad (Button). Lo primero que debemos hacer es meter las imágenes del archivo resources, que os adjuntamos en la entrada, a la carpeta /res/drawable-hdpi/. Vamos ahora al fichero que define la interfaz de nuestra actividad Main activity\_main.xml y lo dejamos con esta pinta:

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context="com.example.forevertucode.wifi\_scanners\_netword.MainActivity">

<LinearLayout android:orientation="vertical" android:id="@+id/linearLayout1" android:layout\_height="fill\_parent" android:layout\_width="fill\_parent" android:gravity="center">

<TextView android:id="@+id/textView1" android:textAppearance="?android:attr/textAppearanceLarge" android:text="Escanear Wifi" android:layout\_height="wrap\_content" android:layout\_width="wrap\_content"

android:layout\_gravity="center\_horizontal"

android:textColor="#FF5C42"

></TextView>

<Button android:layout\_height="wrap\_content" android:id="@+id/button1" android:text="" android:layout\_width="wrap\_content" android:background="@drawable/wifii"

android:layout\_gravity="center\_horizontal"></Button>

</LinearLayout>

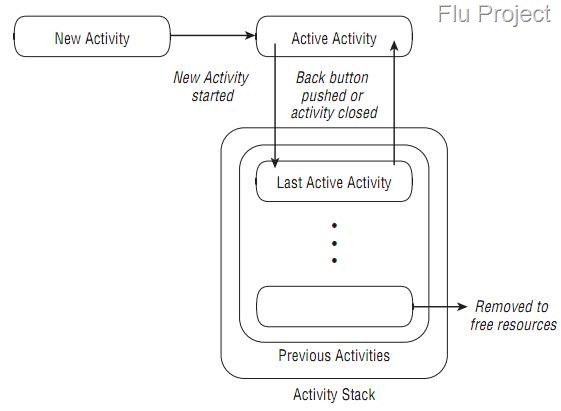
</RelativeLayout>

Como podemos ver en el código del activity\_main.xml, tenemos un LinearLayout principal que contendrá la imagen del banner de fluproject y otro LinearLayout.

El segundo LinearLayout contendrá un TextView con el texto "Escanea Wifi" y un Button con forma de imagen. El motivo de separar la interfaz en dos LinearLayout es conseguir darle gravedad sólo al segundo LinearLayout (android: gravity="center") de forma que sólo se centren en la pantalla el TextView y el Button. Una vez definida la interfaz, pasamos ahora a darle lógica.

## **Definiendo la funcionalidad de la actividad activity\_main.xml**

La funcionalidad de esta actividad es bastante sencilla, exactamente igual que la que utilizamos el otro día para nuestra primera aplicación. La única diferencia es que aquí, además de mostrar un mensaje de texto, crearemos una nueva actividad y la mostraremos. A modo de recordatorio, las actividades (Activity) se corresponden con las pantallas que tendrá nuestra aplicación. En la navegación por pantallas, Android trata las actividades mediante una pila de actividades, de forma que siempre estaremos visualizando la actividad superior de dicha pil, la que se encuentre en la cima. Cuando arranca una nueva actividad siempre se pone en la cima de la pila, en primer plano. Cuando pulsamos el botón 'atrás' estaremos realizando una acción equivalente a 'desapilar' , cerraremos la actividad actual y mostraremos la actividad que esté en la cima de la pila. Si no hay ninguna actividad más, saldremos de la aplicación.

[](https://github.com/fercho0/Wifi_Scanners_Netword/blob/master/img/w1.jpg)

Volviendo a la lógica de la actividad activity\_main, tendremos una clase MainActivity.java con esta apariencia:

import android.content.Intent;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements View.OnClickListener{

/\*\* Called when the activity is first created. \*/

private Button button\_scan;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

this.button\_scan = (Button)findViewById(R.id.button1);

this.button\_scan.setOnClickListener(this);

}

@Override

public void onClick(View v) {

// TODO Auto-generated method stub

switch (v.getId()){

case (R.id.button1):

Toast.makeText(this, "Escaneando redes wifi...", Toast.LENGTH\_LONG).show();

Intent wifiList = new Intent(this, WifiList.class);

startActivity(wifiList);

break;

}

}

}

Como vemos implementamos de OnClickListener para escuchar eventos del Button que hemos definido para escanear. En el método onClick vemos que al pulsar el Button (con id button1 en nuestro main.xml) sacamos una alerta textual por pantalla con contenido "Escaneando redes wifi..." y que además creamos una nueva actividad por medio de la clase Intent:

[Intent wifiList = new Intent(this, Wifilist.class);startActivity(wifiList);](https://github.com/fercho0/Wifi_Scanners_Netword/blob/master)

Al constructor de la clase Intent, clase responsable de crear la nueva actividad, le indicamos la actividad origen y la actividad destino que queremos que se muestre a continuación, que será la actividad Wifilist, la cual todavía no hemos definido ni como actividad ni como clase. Para definir la actividad Wifilist tenemos que crearnos una nueva actividad que se compondrá de una clase Wifilist.java y de un fichero xml wifilist.xml (por convenio de nombres vamos a intentar siempre llamar a la clase de la actividad y a su fichero xml igual, salvo que la primera letra del nombre de la clase será mayúscula y la primera letra del nombre del fichero xml será minúscula):

- Creamos una nueva clase en nuestro paquete com.example.forevertucode.wifi\_scanners\_netword de la siguiente forma: Botón derecho del ratón sobre el paquete, New - > Class, en SuperClass escribimos android.app.Activity , le damos un nombre Wifilist y a aceptar. Una vez creada, pinchamos con el botón derecho del ratón sobre la clase, Source -> Override/Implement Methods y seleccionamos el método onCreate(Bundle)

- Una vez creada la clase creamos su fichero wifilist.xml, para ello nos vamos a la carpeta /res/layout pulsamos el botón derecho del ratón y seleccionamos New->Android XML File, le damos como nombre wifilist y finalizam

- Una vez creada la actividad debemos definirla en el fichero AndroidManifest.xml. Abrimos dicho fichero, nos vamos al apartado Application y bajamos a la sección Application Nodes. Ahí podemos observar como tenemos solamente una actividad declarada, la actividad Main. Para declarar la actividad Wifilist pulsamos Add... -> Activity y rellenamos sus propiedades a la derecha. Sólo pondremos el nombre de la actividad Wifilist y comprobamos que se cambia en el cajetín de Application Nodes.

## **Diseñando la interfaz de la actividad Wifilist**

Como pudimos observar en la imagen del principio, nuestra segunda pantalla tendrá una lista con todas las redes wifi que nuestro móvil haya encontrado. Luego el fichero xml que define la actividad Wifilist wifilist.xml tendrá la siguiente apariencia

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content">

<ListView android:layout\_height="match\_parent" android:id="@+id/listView1" android:layout\_width="match\_parent"></ListView>

</LinearLayout>

Como podemos observar tiene un LinearLayout con un único componente ListView.

Cada elemento de la lista también debe tener un aspecto, en nuestro caso no van a ser elementos simples, como podemos observar en la imagen del principio tendremos más de un elemento en cada item de la lista (ssid y seguridad), además estos elementos varían en tamaño y en color luego habrá que hacer otro proceso de definición antes de ponernos con la lógica. Este proceso pasa por definirnos otro fichero xml, de la misma forma que nos definimos el anterior, que contendrá el aspecto de cada item de la lista, el fichero se llamará elementitems.xml y tendrá este aspecto:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView android:id="@+id/strssid"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:textStyle="bold"

android:textSize="40px"

android:textColor="#E74C3C" />

<TextView android:id="@+id/strsecurity"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:textStyle="normal"

android:textSize="24px"

android:textColor="#2C3E50" />

</LinearLayout>

Como podemos ver, disponemos de un LinearLayout que alberga dos TextView, el primero hace referencia al ssid de la red y el segundo a la seguridad que tiene configurada. Y con esto ya tendríamos diseñada la interfaz de nuestra actividad Wifilist.

## **Definiendo la funcionalidad de la actividad Wifilist:**

Lo primero que vamos a hacer para definir la funcionalidad de Wifilist es crearnos una clase básica en Java llamada [Element](https://github.com/fercho0/Wifi_Scanners_Netword/blob/master). Como hemos dicho anteriormente nuestra lista no va a contener items simples sino que contendrá items compuestos por dos TextView, esto debemos modelarlo en una clase. La clase Element estará ubicada en el paquete com.example.forevertucode.wifi\_scanners\_netword y tendrá la siguiente apariencia:

public class Element {

private String title;

private String subtitle;

public Element(String t, String s){

this.title=t;

this.subtitle=s;

}

public String getTitle(){

return this.title;

}

public String getSubtitle(){

return this.subtitle;

}

}

En la clase Element tendremos dos atributos uno será title que hará referencia en un futuro al SSID de la red, y el otro subtitle que hará referencia a la seguridad de la red. Además tendremos dos métodos para obtener tanto el atributo title como el subtitle.

A continuación, vamos a intentar explicar el código que debe tener la clase Wifilist.java en tres partes para intentar aclarar la funcionalidad.

import android.app.Activity;

import android.content.Context;

import android.net.wifi.ScanResult;

import android.net.wifi.WifiManager;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.TextView;

import java.util.List;

public class WifiList extends AppCompatActivity {

private Element[] nets;

private WifiManager manWifi;

private List<ScanResult> wifiList;

public void onCreate(Bundle savedInstanceState){

super.onCreate(savedInstanceState);

//Configura escaner.

this.manWifi = (WifiManager) getSystemService(Context.WIFI\_SERVICE);

this.manWifi.startScan();

this.wifiList = this.manWifi.getScanResults();

//Configura lista adaptadora de wifis.

this.nets = new Element[wifiList.size()];

for(int i = 0; i < wifiList.size(); i++){

String item = wifiList.get(i).toString();

String[] vector\_item = item.split(",");

String item\_essid = vector\_item[0];

String item\_capabilities = vector\_item[2];

String ssid = item\_essid.split(":")[1];

String security = item\_capabilities.split(":")[1];

nets[i]=new Element(ssid, security);

}

//Configura lista final de wifis.

setContentView(R.layout.wifilist);

AdapterElements adapter = new AdapterElements(this);

ListView netlist = (ListView)findViewById(R.id.listView1);

netlist.setAdapter(adapter);

}

class AdapterElements extends ArrayAdapter<Object> {

Activity context;

public AdapterElements(Activity context) {

super(context, R.layout.elementitems, nets);

this.context = context;

}

public View getView(int position, View convertView, ViewGroup parent) {

LayoutInflater inflater = context.getLayoutInflater();

View item = inflater.inflate(R.layout.elementitems, null);

//Carga SSID.

TextView lblTitle = (TextView)item.findViewById(R.id.strssid);

lblTitle.setText(nets[position].getTitle());

//Carga Security.

TextView lblSubtitle = (TextView)item.findViewById(R.id.strsecurity);

lblSubtitle.setText(nets[position].getSubtitle());

return(item);

}

}

}

Empezaremos nuestro código Wifilist.java definiendo tres atributos de clase: nets -> array de objetos Element que se encargará de guardar las redes descubiertas y posteriormente pasarle la información a un ArrayAdapter para que la lista pueda ser construida; manWifi -> Objeto de la clase WifiManager que nos permitirá acceder a la interfaz Wifi de nuestro dispositivo móvil y utilizar su funcionalidad; wifiList -> Objeto de la clase List donde guardaremos los Strings en bruto resultado de un escáner wifi.

**Punto 1:** En este punto llamamos a la interfaz wifi de nuestro dispositivo móvil y guardamos el resultado en el atributo manWifi. Posteriormente realizamos el escáner de redes mediante el método startScan () de la clase WifiManager y por último guardamos el resultado en bruto en nuestro atributo wifiList por medio del método getScanResults().

**Punto 2:** En este punto vamos a parsear la información que nos interesa del resultado del escáner guardado en bruto en el atributo wifiList. Recorreremos la lista wifiList, rescataremos la información que nos interesa de cada red (ssid y security) e iremos creando objetos Element y guardándolos en nuestro atributo nets. Una vez parseada y guardada toda la información resultante del escáner cargaremos a nuestra actividad la interfaz correspondiente wifilist.xml. Después crearemos un array adaptador AdapterElements de objetos Elements para nuestra lista que definiremos en el Punto 3.

**Punto 3**: Nos definimos una clase llamada AdapterElements que herede de ArrayAdapter. Esta clase nos va a servir para construir nuestra lista a partir del atributo nets. Esta clase tendrá un sólo atributo context que indicará el contexto donde se va a construir la lista, en la actividad presente. Como podemos observar AdapterElements tiene un constructor de clase que llama a su clase padre ArrayAdapter y un método getView. El método getView es el que se encarga de visualizar los elementos del atributo nets en forma de ListView. getView será llamado tantas veces como posiciones tenga nuestro atributo nets, además getView nos proporciona un parámetro position que nos servirá de índice para obtener toda la información de nets. Dentro de getView formamos nuestro item cargando el fichero elementitems.xml mediante el método inflate de la clase LayoutInflate, después sólo manipulamos el contenido textual de los TextView en función del valor de nets y listo!

Ya tenemos nuestra aplicación completamente diseñada, sólo nos queda un pequeño detalle, decirle al AndroidManifest.xml que nos deje utilizar el wifi de nuestro dispositivo móvil. Para ello nos vamos al fichero y añadimos los siguientes permisos tal y como explicamos en la entrada anterior:

[android.permission.ACCESS\_WIFI\_STATEandroid.permission.CHANGE\_WIFI\_STATE](https://github.com/fercho0/Wifi_Scanners_Netword/blob/master)