

Desarrollo de aplicación en Android con acceso a Web Service

Introducción al desarrollo de aplicaciones móviles en
Android.

Felipe Lima Cortés

Noviembre de 2012



Contenido

Acerca de la aplicación: Contactos App	3
Arquitectura de la aplicación: BD	3
Web Service al que se conectará la aplicación	3
Desarrollo de la aplicación: Contactos App.....	4
Crear el proyecto.....	4
Estructura de la aplicación	9
Diseño de la interfaz de usuario.....	10
Creando el Encabezado	11
Agregando el encabezado	14
Probando la aplicación en el emulador.....	15
Cambiar la apariencia de nuestra interfaz.....	16
Apariencia de los textos.....	18
Quitar barra de título de la aplicación	19
Preparando los espacios de nombre (subpaquetes).....	19
Creamos el modelo	21
Manipulando los elementos de la interfaz.....	22
Manipulando el ListView de nuestro layout	23
Creando un Adaptador.....	25
Identificador de Elementos (ID de contacto)	31
Diseño de layout y creación de Activities: Registrar y Detalles	32
Layout para agregar nuevo Contacto.....	32
Layout para detalles de Contacto	35
Crear Activity AgregarContactoActivity.....	36
Crear Activity DetallesContactoActivity	38
Controlar eventos de los botones	38
Registrar los Activities	39
Probando el resultado logrado hasta el momento	40
Programar los botones en AgregarContactoActivity	40
Controlar el evento al seleccionar un elemento de la lista de Contactos.....	42
Pasar datos entre Activities.....	43
Conectar la aplicación al Web Service	46

La librería ksoap2-android.....	47
Creando las clases necesarias para el acceso a datos.....	48
Agregar permisos de red a la aplicación	57
Manipulando los datos a través del Web Service	57
El evento onResume()	78
El Uso de Intents disponibles en Android	78
El marcado telefónico	78
El botón email	80
Corriendo la aplicación en Android 4.0.3.....	81
Implementando Hilos o Threads para el acceso a través de la red	81
Adaptando las llamadas al WS a los nuevos Threads.....	83
El resultado en Android ICS 4.0.3.....	84
Últimos detalles en la aplicación.....	87
Ocultar elementos de la UI.....	87
Mensajes de confirmación	88
Recursos utilizados en el proyecto.....	90
Conclusión.....	90

Acerca de la aplicación: Contactos App

La aplicación a implementar está planeada para poner en práctica el desarrollo de aplicaciones básicas en Android que comprende el diseño de interfaces de usuario, navegación entre ventanas, manipulación de componentes de la interfaz, y el consumo de servicios web (Web Services).

Como parte del aprendizaje, agregaremos a la aplicación las partes necesarias para que esta pueda acceder y consumir los datos provenientes de un Web Service (SOAP) disponible en un servidor web Apache Httpd implementado en PHP utilizando el Zend Framework, logrando con ello la interoperabilidad de aplicaciones de software.

Arquitectura de la aplicación: BD

La base de datos alojada en el servidor web solo contará con una única tabla, que se muestra a continuación:

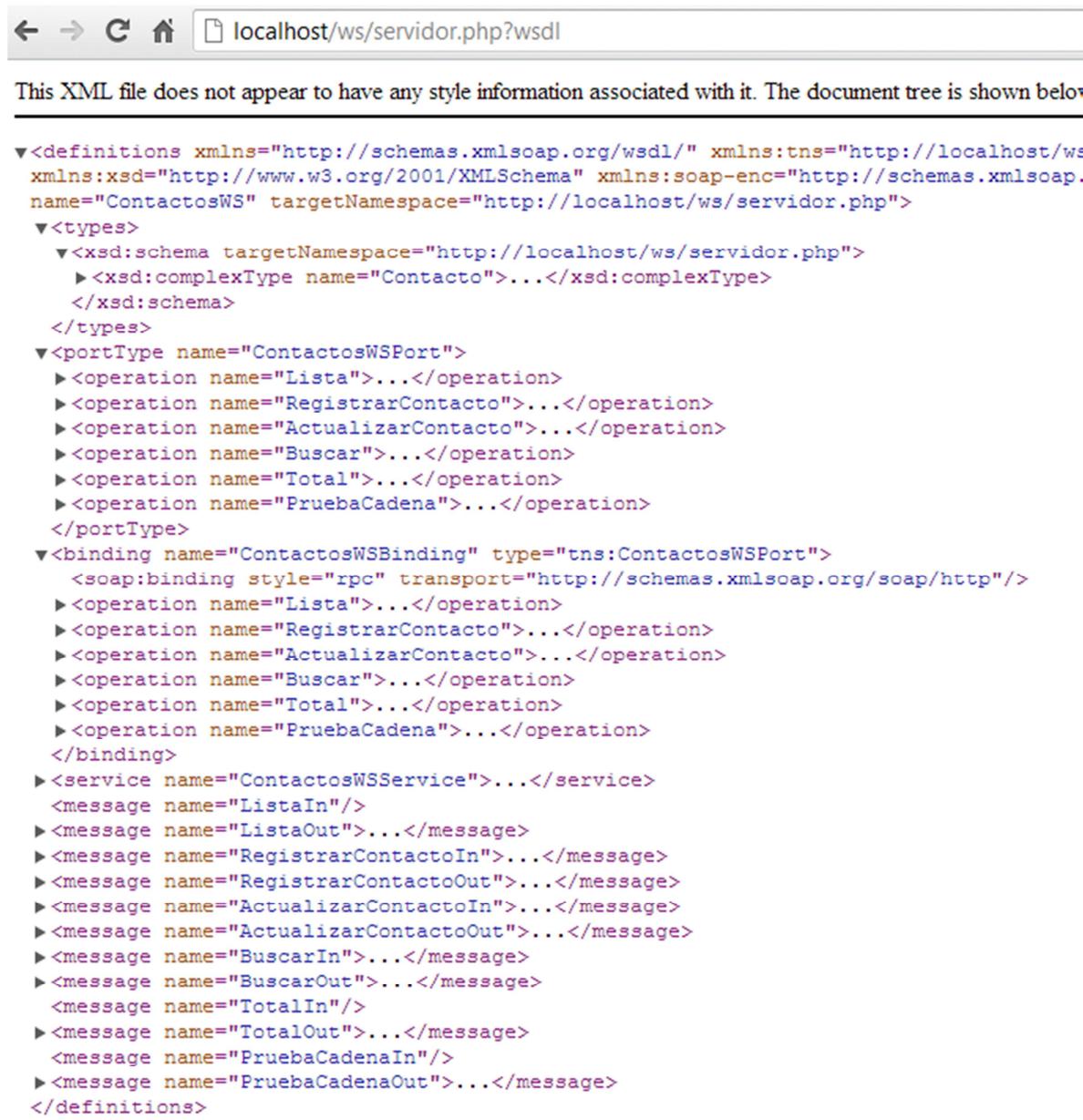
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
1	<u>id</u>	int(11)			No	Ninguna	AUTO_INCREMENT
2	<u>nombre</u>	varchar(24)	utf8mb4_unicode_ci		No	Ninguna	
3	<u>apellidos</u>	varchar(24)	utf8mb4_unicode_ci		No	Ninguna	
4	<u>email</u>	varchar(32)	utf8mb4_unicode_ci		Sí	NULL	
5	<u>telefono</u>	varchar(24)	utf8mb4_unicode_ci		Sí	NULL	

↑ Marcar todos / Desmarcar todos Para los elementos que están marcados: Examinar Ca

Ilustración 1.- Tabla que almacena los datos del WS.

Web Service al que se conectará la aplicación

El Web Service al cual se conectará la aplicación a desarrollar es el que se aprecia en la imagen.



This XML file does not appear to have any style information associated with it. The document tree is shown below:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://localhost/ws" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding" name="ContactosWS" targetNamespace="http://localhost/ws/servidor.php">
  <types>
    <xsd:schema targetNamespace="http://localhost/ws/servidor.php">
      <xsd:complexType name="Contacto">...</xsd:complexType>
    </xsd:schema>
  </types>
  <portType name="ContactosWSPort">
    <operation name="Lista">...</operation>
    <operation name="RegistrarContacto">...</operation>
    <operation name="ActualizarContacto">...</operation>
    <operation name="Buscar">...</operation>
    <operation name="Total">...</operation>
    <operation name="PruebaCadena">...</operation>
  </portType>
  <binding name="ContactosWSBinding" type="tns:ContactosWSPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Lista">...</operation>
    <operation name="RegistrarContacto">...</operation>
    <operation name="ActualizarContacto">...</operation>
    <operation name="Buscar">...</operation>
    <operation name="Total">...</operation>
    <operation name="PruebaCadena">...</operation>
  </binding>
  <service name="ContactosWSService">...</service>
    <message name="ListaIn"/>
    <message name="ListaOut">...</message>
    <message name="RegistrarContactoIn">...</message>
    <message name="RegistrarContactoOut">...</message>
    <message name="ActualizarContactoIn">...</message>
    <message name="ActualizarContactoOut">...</message>
    <message name="BuscarIn">...</message>
    <message name="BuscarOut">...</message>
    <message name="TotalIn"/>
    <message name="TotalOut">...</message>
    <message name="PruebaCadenaIn"/>
    <message name="PruebaCadenaOut">...</message>
  </service>
</definitions>
```

Ilustración 2.- Vista del web Service

Desarrollo de la aplicación: Contactos App

Crear el proyecto

Configurado el entorno de desarrollo (Eclipse IDE) con las herramientas necesarias para el desarrollo, vamos a archivo>> Nuevo>> Proyecto donde seleccionamos *Android Application Project*

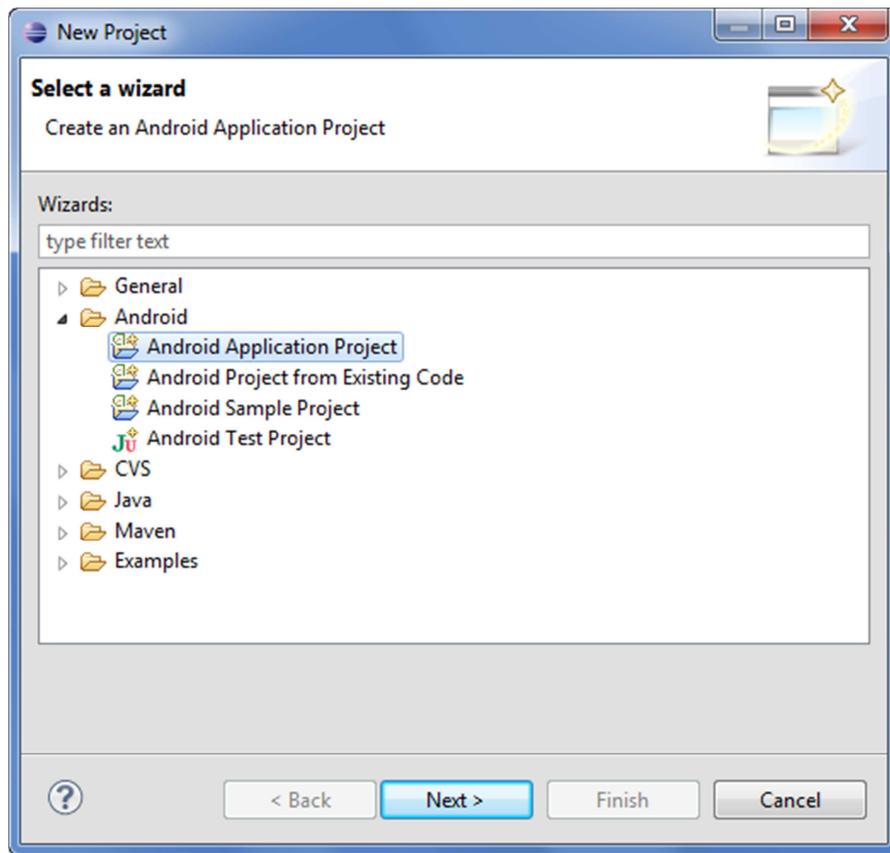


Ilustración 3.- Asistente para crear nuevo proyecto

Establecemos el nombre del proyecto, Nombre de la aplicación, y el nombre del paquete principal de la aplicación; seleccionamos también la versión de Android para la cual se desarrollará la aplicación, así como la versión mínima de Android disponible en el dispositivo en el cual se pretenda instalar la aplicación.

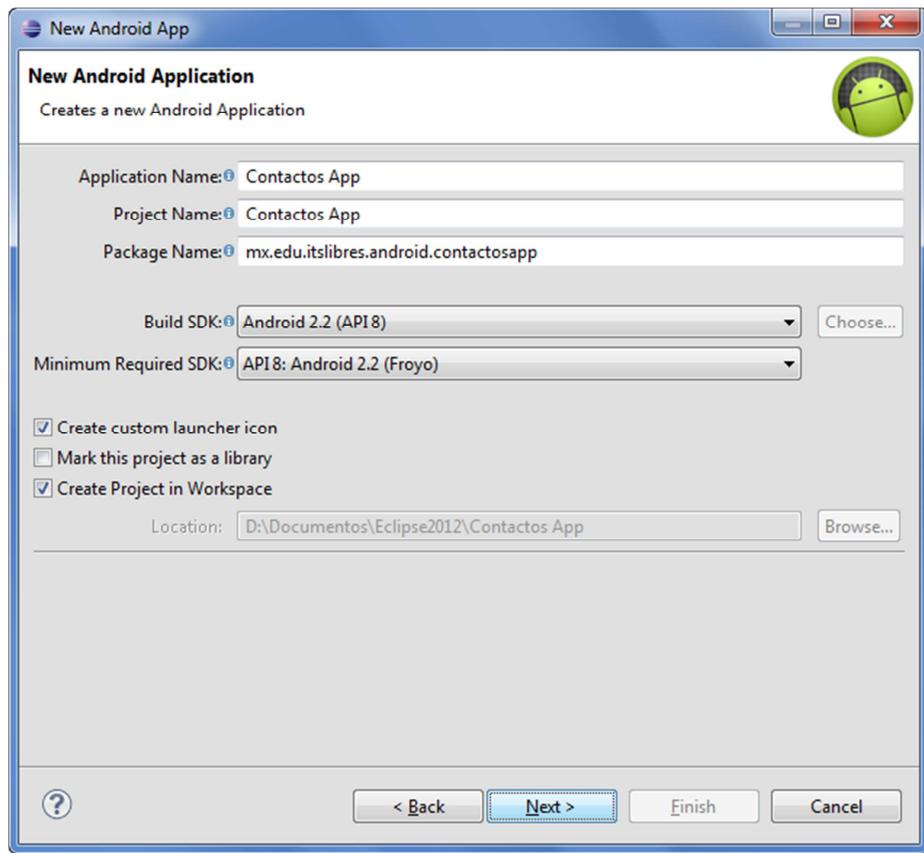


Ilustración 4.- Asistente para crear nuevo proyecto

Nota: El nombre del paquete se definió como **mx.edu.itslibres.android.contactosapp** haciendo referencia a: aplicación llamada ContactosApp que forma parte del grupo de aplicaciones Android del Instituto Tecnológico Superior de Libres (cuyo dominio es itslibres.edu.mx).

Al activarse “Create custom launcher icon”, en la siguiente ventana se creará el ícono del lanzador de la aplicación “launcher”.

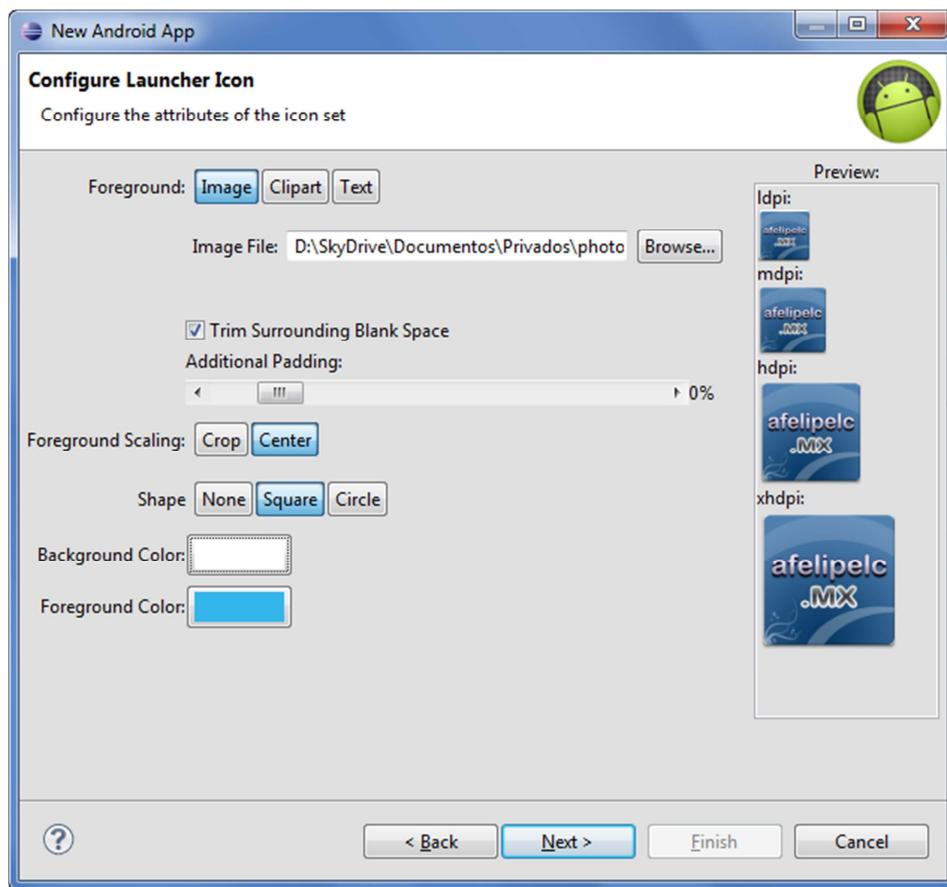


Ilustración 5.- Asistente para crear nuevo proyecto

El asistente pregunta si se desea crear una actividad (Ventana).

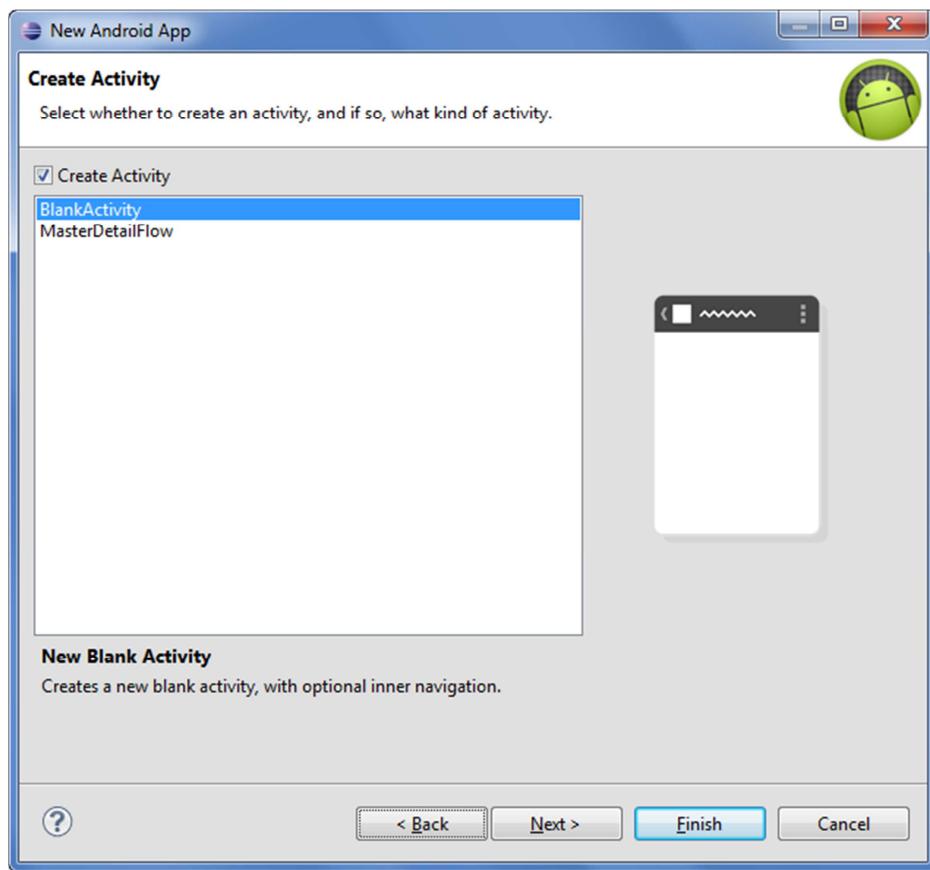


Ilustración 6.- Asistente para crear nuevo proyecto

Si se elige crear una actividad principal como punto de entrada de la aplicación “Main Activity”, deben establecerse las propiedades de la actividad.

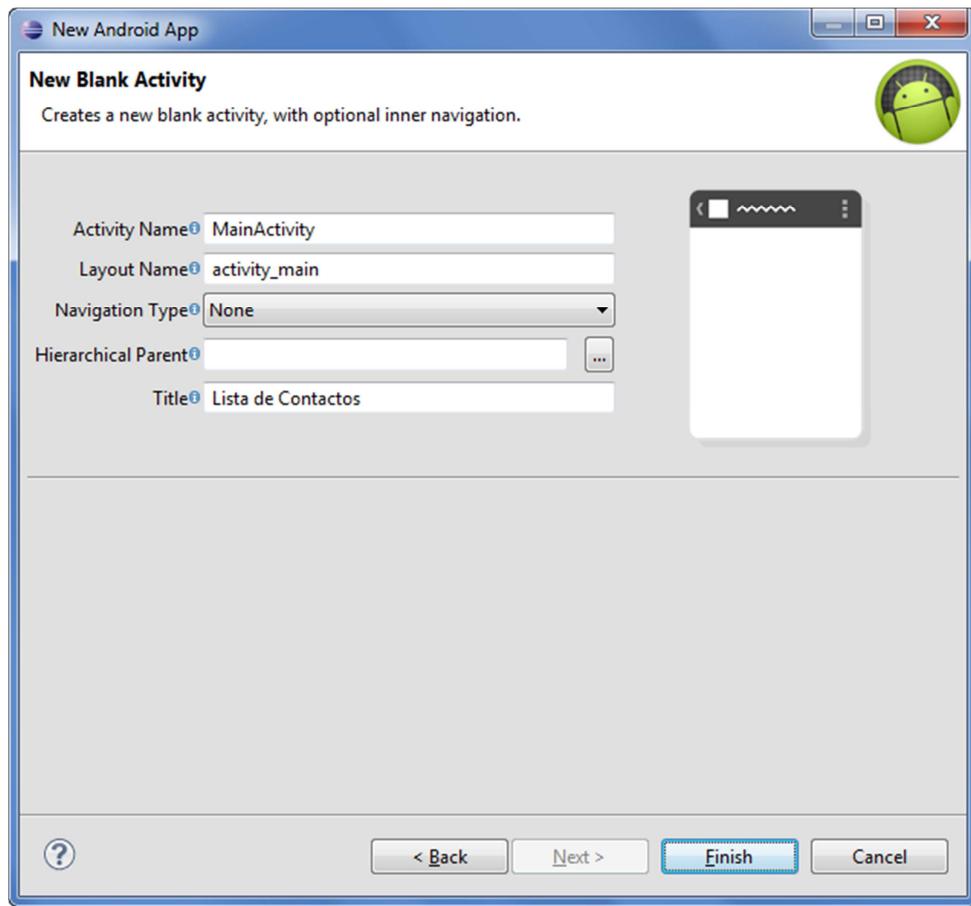


Ilustración 7.- Asistente para crear nuevo proyecto

Estructura de la aplicación

Al hacer clic en Finish, el proyecto será creado dando como resultado la siguiente estructura.

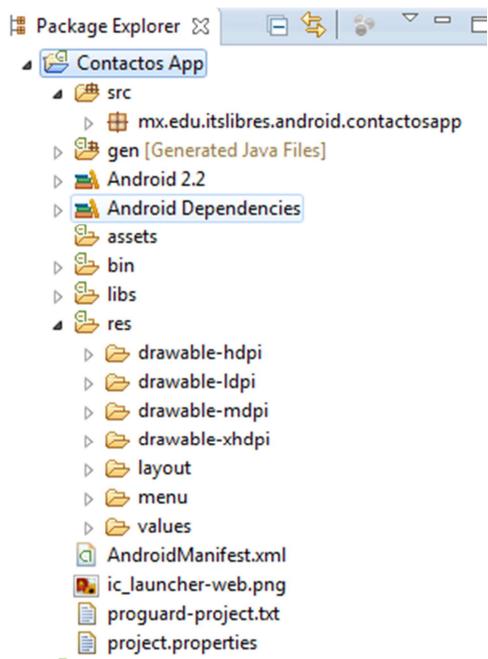


Ilustración 8.- Estructura del proyecto

Diseño de la interfaz de usuario

Buscar patrones que nos ayuden a realizar un buen diseño de nuestras aplicaciones, podemos encontrar muchos en <http://www.androidpatterns.com/> y <http://www.android-app-patterns.com/>

La aplicación mostrará la información en forma de lista, así que busquemos patrones que se adapten a nuestras necesidades: <http://www.android-app-patterns.com//category/lists>

Una posible propuesta es esta:

Desarrollo de aplicación en Android con acceso a Web Service

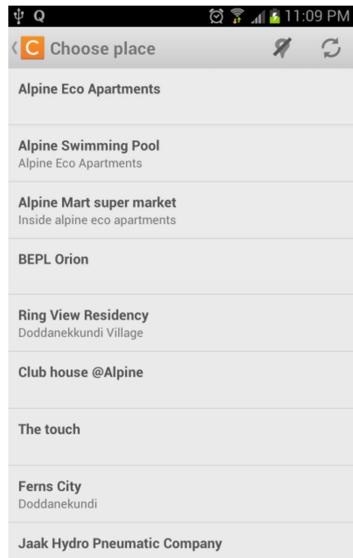


Ilustración 9.- Ejemplo de interfaz de usuario Android

El diseño propuesto se conforma de un encabezado (header), el contenedor principal será una lista, también se mostrarán los detalles del contacto en otra interfaz.

Esto es lo que crea el asistente para crear proyecto Android

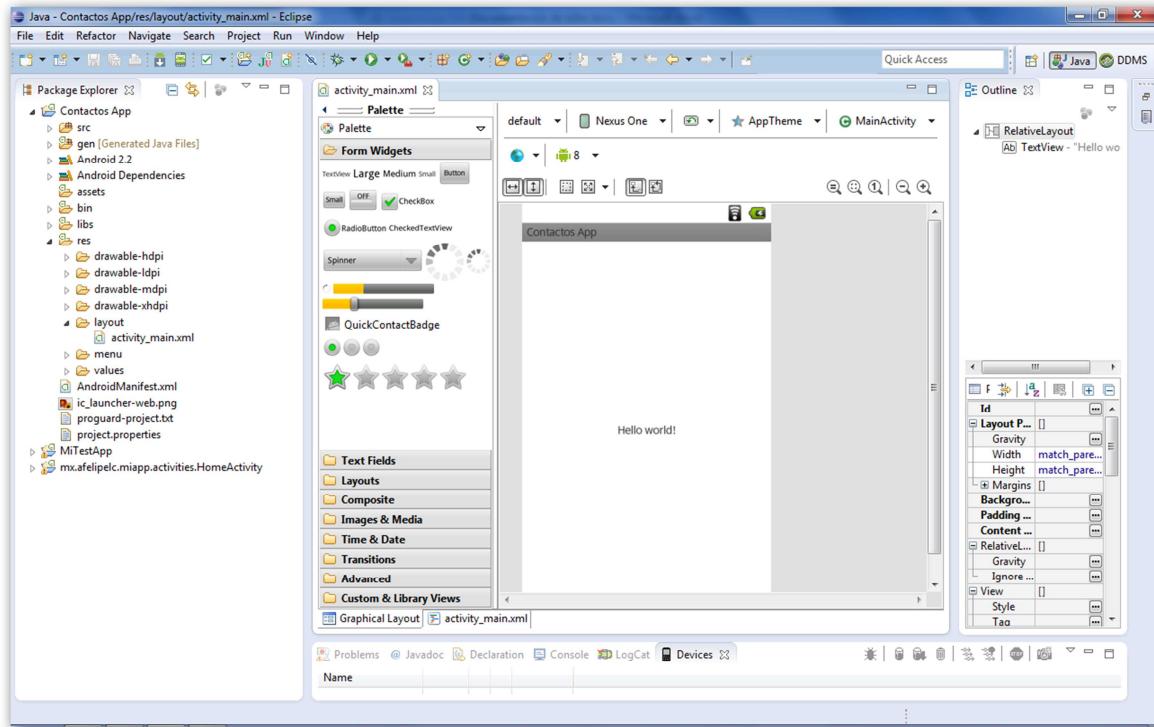


Ilustración 10.- Proyecto Android creado

Creando el Encabezado

Agregamos un nuevo archivo layout

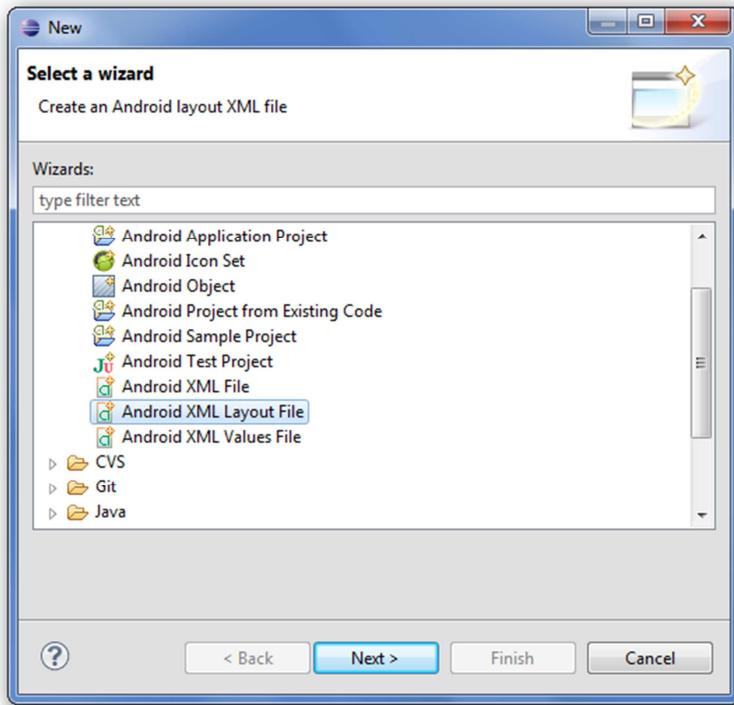


Ilustración 11.- Agregar un nuevo layout

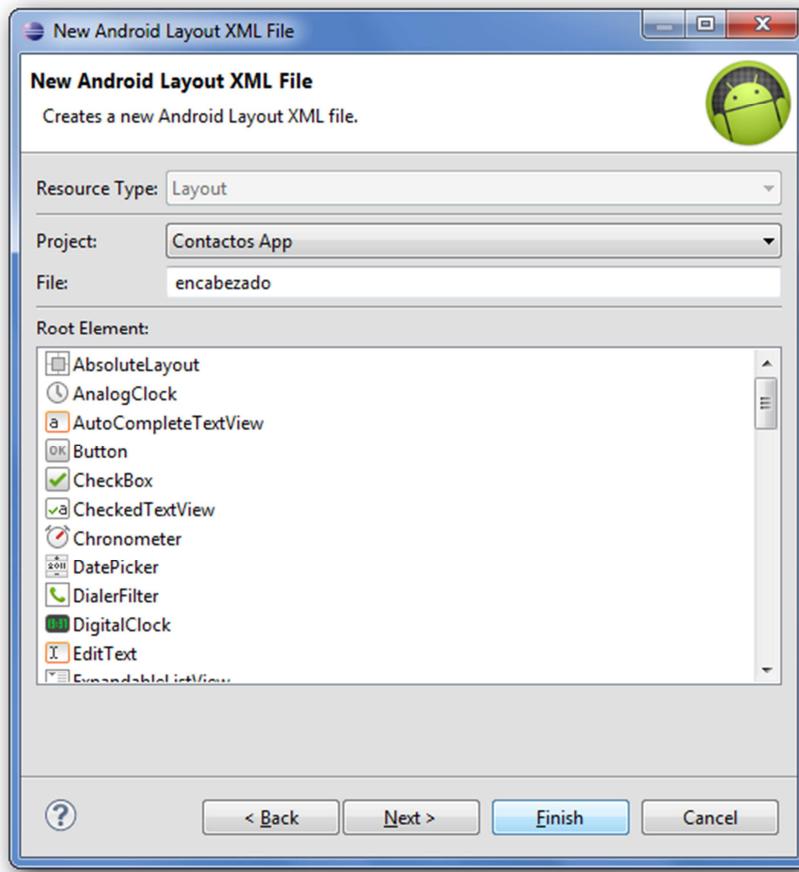


Ilustración 12.- Agregar un nuevo layout

Agregar los elementos como en la imagen: una imagen con el ícono del itsl, un textView y un ImageButton

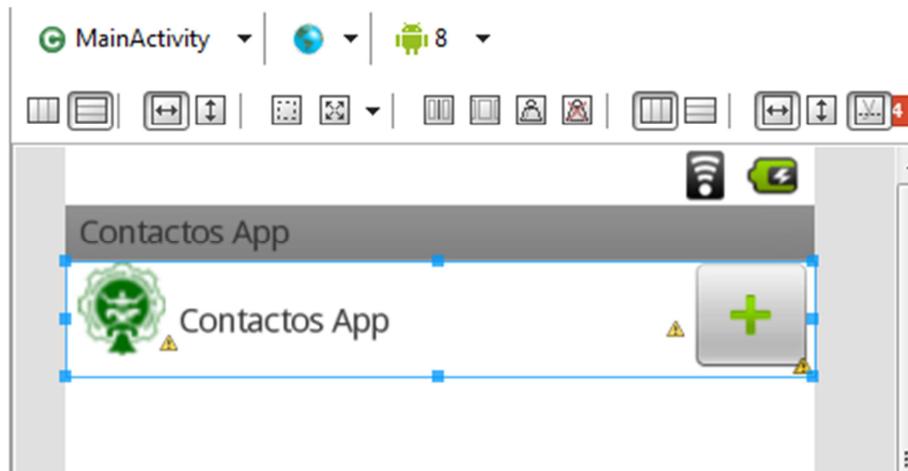


Ilustración 13.- Diseño del encabezado para la aplicación

Donde el TextView tiene la propiedad **weight** en 1 y la imagen con weight en 0.09 para dar un espacio apropiado, el elemento que aparece seleccionado es un linearlayout con orientación horizontal.

En la ventana OutLine observamos cómo va queda la estructura de este layout, donde también se muestran los tipos, su Id y el recurso de cada elemento.

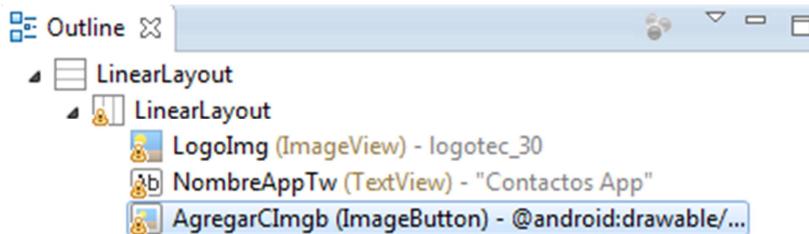


Ilustración 14.- Estructura del encabezado (layout)

Agregando el encabezado

En el Main_Layout que se encuentra vacío, agregamos un nuevo elemento “Include other layout” donde seleccionaremos nuestro layout llamado Encabezado

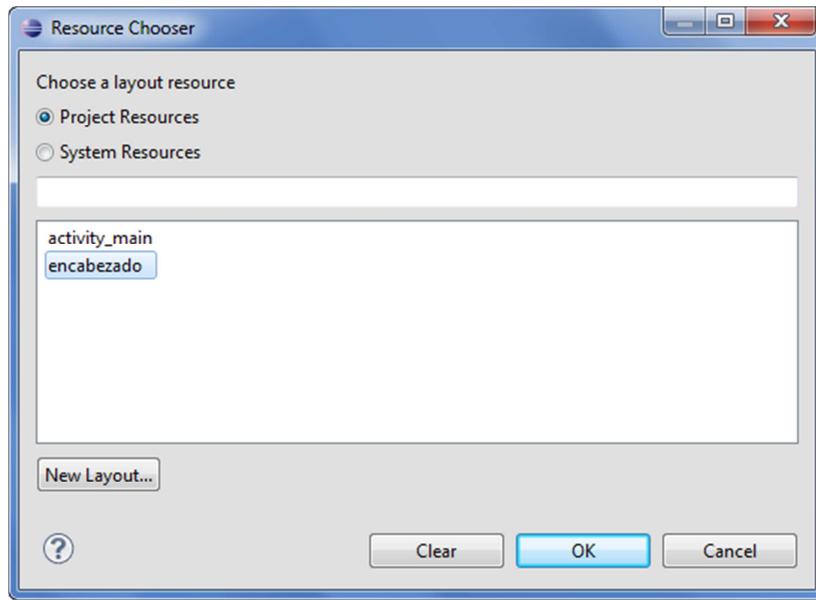


Ilustración 15.- Seleccionando el layout a incluir en otro

Agregado el layout debemos pegarlo a la parte superior

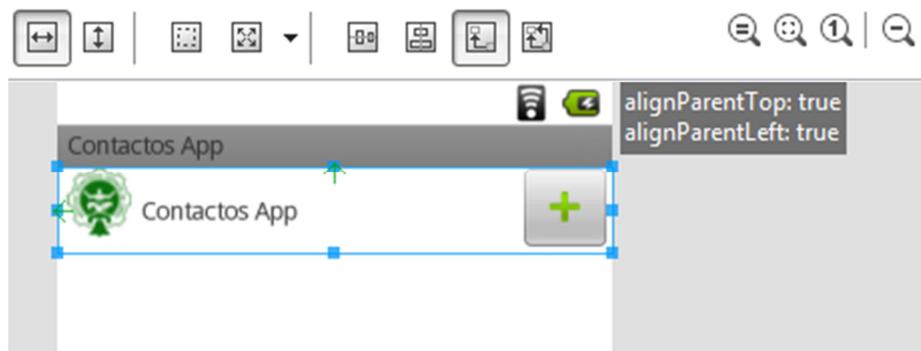


Ilustración 16.- Layout incuido en otro layout

Probando la aplicación en el emulador

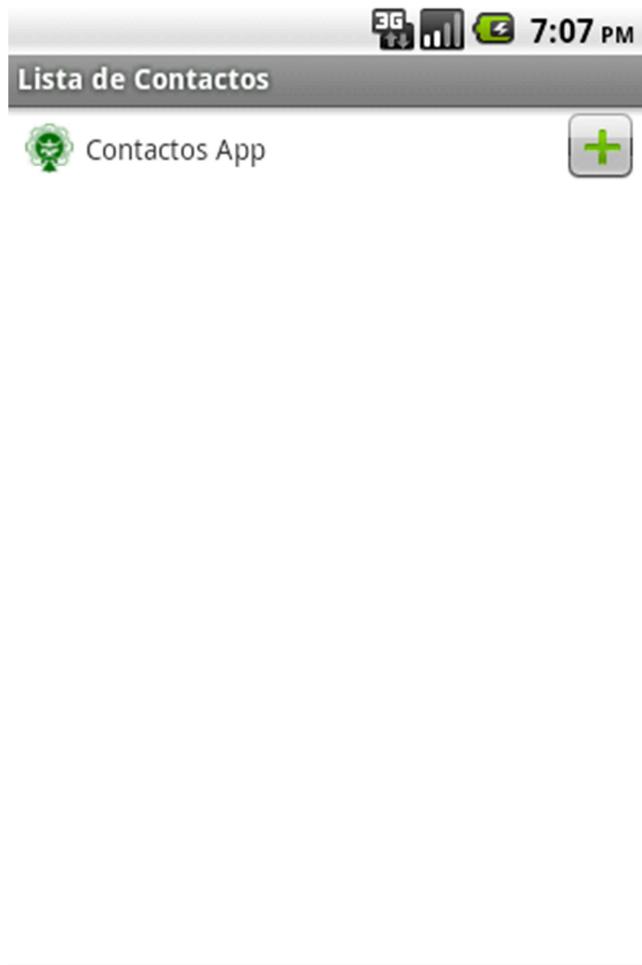


Ilustración 17.- Vista del resultado en el emulador

Cambiar la apariencia de nuestra interfaz.

En la carpeta **res** agregamos una nueva carpeta llamada **drawable**

Dentro de esta última agregamos un nuevo archivo llamado **fondoencabezado.xml** de tipo Android XML, seleccionamos el tipo de recurso, y el elemento principal (Shape, para cambiar la apariencia de los elementos de la interfaz)

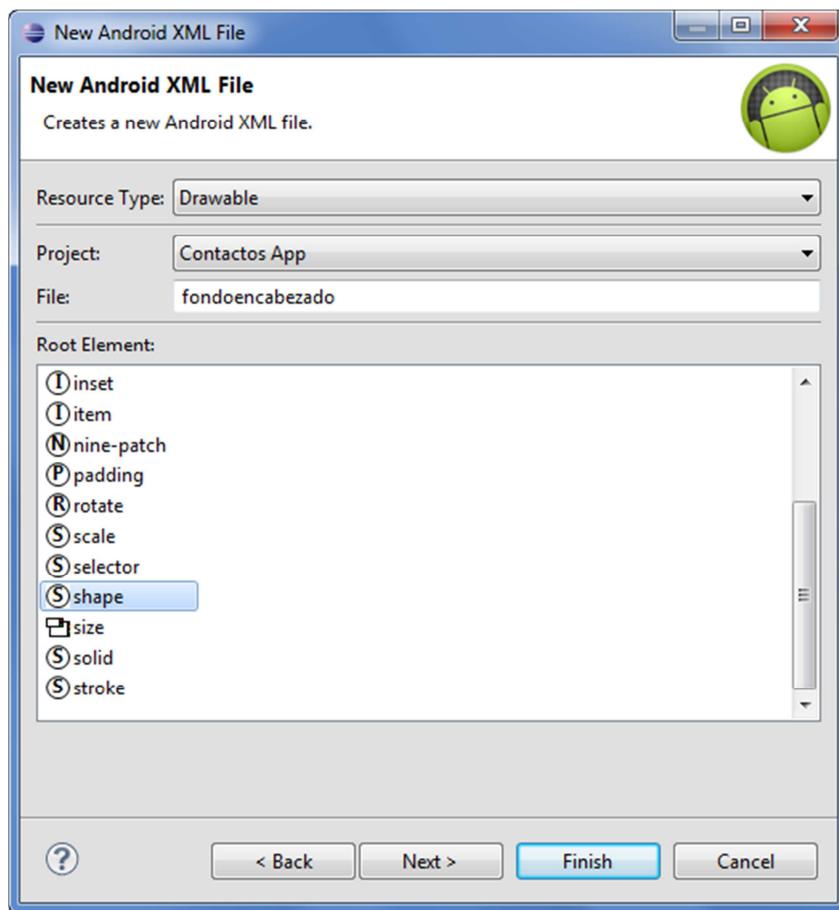


Ilustración 18.- Agregar nuevo recurso:: drawable

Consultar <http://developer.android.com/guide/topics/resources/drawable-resource.html> para obtener la información necesaria acerca de los drawable resources en Android.

Necesitamos un generador de gradients para realizar nuestras mezclas de colores:
<http://gradients.glrzad.com/>

Agregamos un elemento gradient al nuevo archivo xml quedando su código

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
```

```
    android:startColor="#eaeaea"
    android:endColor="#c4c4c4"
    android:angle="-90" />

</shape>
```

O el recurso puede ser creado directamente como Gradient sin necesidad de ser un Shape, pero como Shape se le pueden agregar distintos atributos a los cuales deberá adaptarse el elemento.

Vamos al layout llamado encabezado.xml, seleccionamos el linearlayout que abarca todo el encabezado, en su propiedad background agregamos el nuevo recurso fondoencabezado

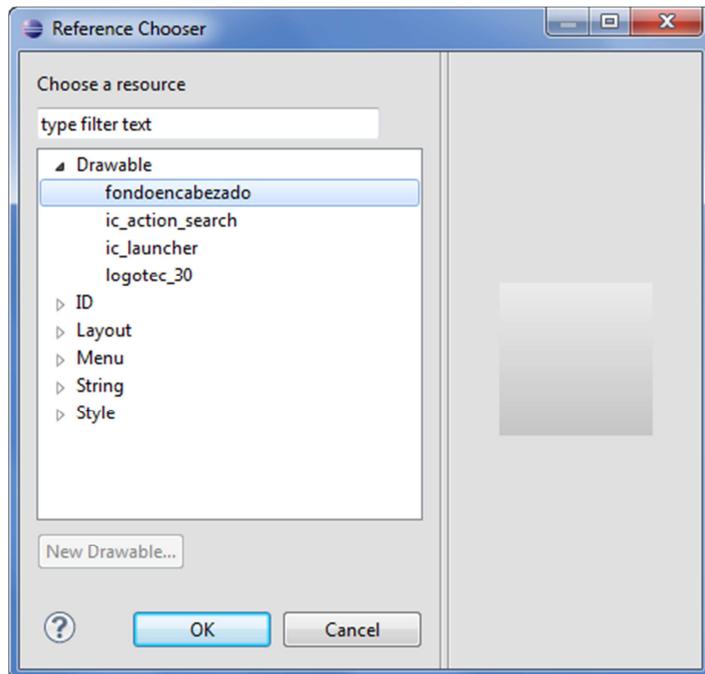


Ilustración 19.- Agregando un recurso drawable como fondo

El fondo ha sido agregado

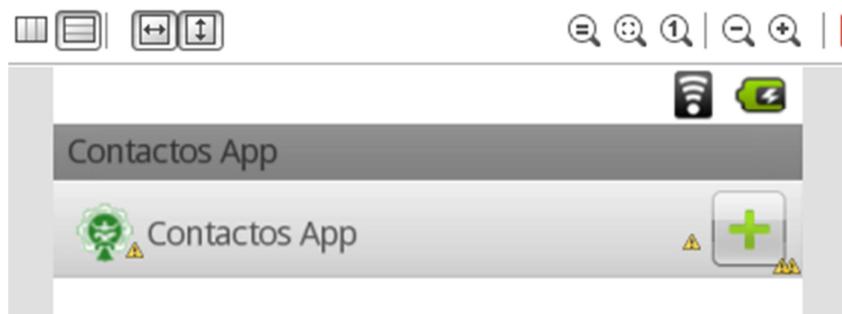


Ilustración 20.- Fondo agregado.

Crear ahora un fondo para el contenido principal, pero antes de aplicarlo agreguemos un linearlayout (en main_layout) debajo del encabezado y dentro de este último layout metemos el Listview, el nuevo fondo lo aplicarlo al linearlayout agregado al final (contenedor del listview).

El diseño tomará forma

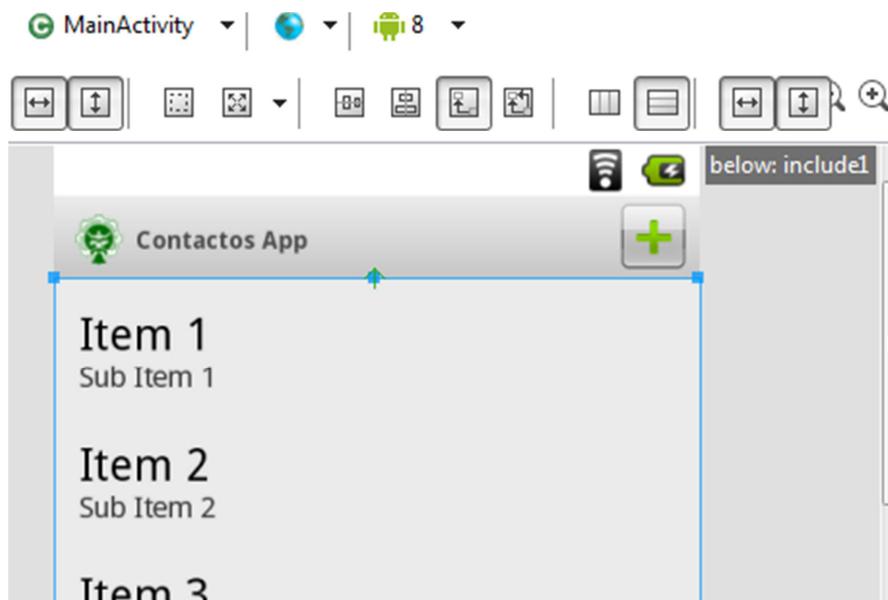


Ilustración 21.- Nuevo fondo agregado al contenedor principal

Apariencia de los textos.

También podemos definir la apariencia del formato que tendrán los textos de los elementos, a esto se le llama estilos, debe agregarse al llamado **styles.xml** dentro de la carpeta **values**.

Agregaremos un estilo llamado estilotitulo, el archivo styles.xml tendrá el siguiente código

```
<resources>
    <style name="AppTheme" parent="android:Theme.Light" />

    <style name="estilotitulo">
        <item name="android:textColor">#4D4B4C</item>
        <item name="android:textSize">14sp</item>
        <item name="android:textStyle">bold</item>
    </style>
</resources>
```

Vamos al layout encabezado, seleccionamos el textview que contiene el título y en su propiedad Style asignamos en nuevo estilo agregado.

Probando la aplicación, este será el resultado



Ilustración 22.- Resultado obtenido en el emulador

Esta es la estructura creada hasta ahora es esta

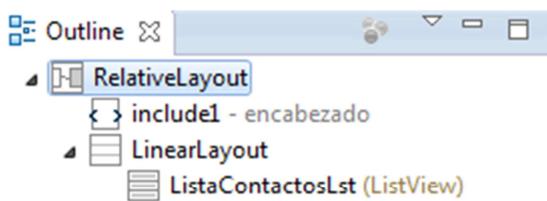


Ilustración 23.- Estructura del activity_main

Quitar barra de título de la aplicación

Para quitar la barra de título a la aplicación, modificar el archivo style.xml editando el tema a

```
<style name="AppTheme" parent="android:Theme.Light.NoTitleBar" />
```

Preparando los espacios de nombre (subpaquetes)

En la aplicación agregaremos diferentes clases que serán necesarias para el funcionamiento de la app, estas clases estarán separadas en grupos o espacios de nombres o paquetes, de forma que haya organización.

Crearemos los siguientes subpaquetes dentro del paquete principal que se encuentra dentro de
src >>> mx.edu.itslibres.android.contactosapp

Los subpaquetes a agregar son:

- **activities**: Contendrá las clases para los Activities (Actividades)
- **models**: Contendrá las clases que conforman el modelo de datos
- **dataaccess**: Contendrá las clases necesarias para el acceso a los datos del web service
- **adapters**: Contendrá las clases encargadas de pasar los datos recibidos de los WS a los elementos ListView

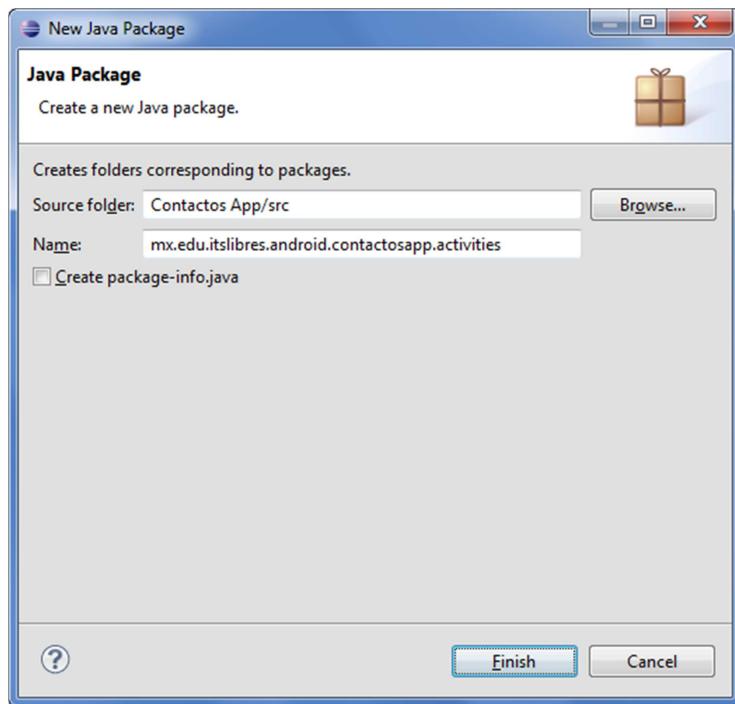


Ilustración 24.- Agregar nuevo paquete

Esta es la estructura nueva de los paquetes

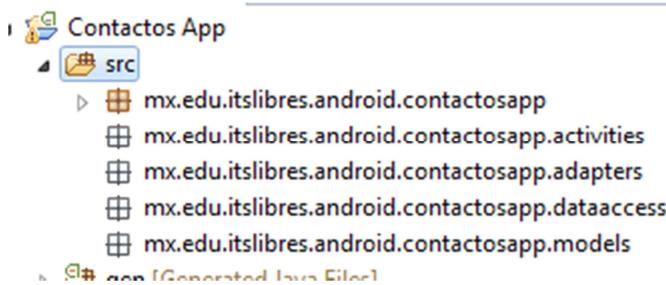


Ilustración 25.- Estructura de los paquetes agregados

El paquete principal contiene un Activity llamado `MainActivity.java`, esta clase debe ser movida al paquete correspondiente (solo arrastrar y soltar; preguntará si se quiere actualizar las referencias, decimos que sí).

Creamos el modelo

Al principio se mostró la estructura de la tabla de la BD, tomaremos como referencia la misma estructura para crear nuestra clase modelo.

Agregamos entonces una nueva clase Java llamada **Contacto** al paquete **models**.

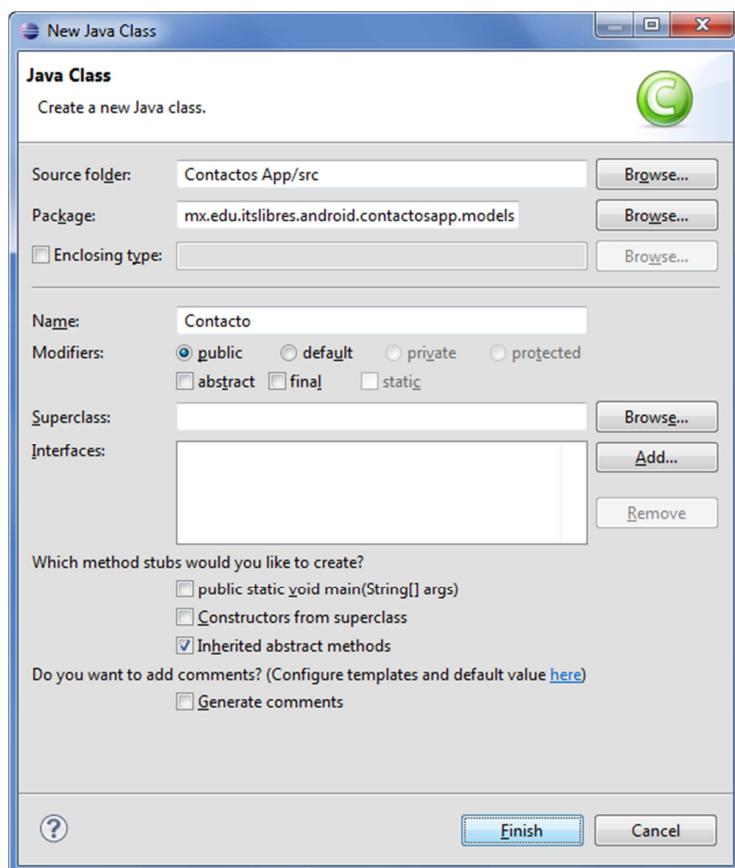


Ilustración 26.- Agregar nueva clase Java

El esquema es: Contacto{integer id; string nombre, string apellidos, string email, string telefono}, implementar sus getters & setters

```
package mx.edu.itslibres.android.contactosapp.models;

public class Contacto {
    public int id;
    public String nombre;
    String apellidos;
    String email;
    String telefono;
    /*
     * Getters & setters here
     */
}
```

Manipulando los elementos de la interfaz.

Revisar la documentación en <http://developer.android.com/reference/android/view/package-summary.html>

Para nuestro proyecto, el punto de entrada es la actividad (clase) llamada MainActivity, este es su código fuente:

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import mx.edu.itslibres.android.contactosapp.R.layout;
import mx.edu.itslibres.android.contactosapp.R.menu;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

La instrucción ***setContentView(R.layout.activity_main);*** define el layout a mostrar para esta actividad.

Para manipular los elementos del layout correspondiente, es necesario crear instancias de objetos del mismo tipo que los elementos, recordemos que cada elemento de la vista se llama View:

Vistas Básicas

- Botones (Button)
- Botones con imagen (ImageButton)
- Etiquetas (TextView)
- Cuadros de texto (EditText).
- Listas (ListView)

- Layouts (View) [todos los elementos anteriores derivan de la clase View]

Sintaxis para recuperar el objeto a manipular.

```
Tipo_Vista nombreobjeto = (Tipo_Vista) findViewById(R.id.Id_del_elemento);
```

ejemplos:

```
Button atrasbtn = (Button) findViewById(R.id.AtrasBtn);
TextView titulo = (TextView) findViewById(R.id.TituloNavbar);
EditText nombretxt = (EditText) findViewById(R.id.NombreClienteTxt);
ListView listaclientes = (ListView) findViewById(R.id.listaclientes);
```

Manipulando el ListView de nuestro layout

Abrimos el activity llamado MainActivity, en el cual agregaremos el código de prueba para mostrar cómo se infla un listview, el código de este activity quedará así

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {

    //declaramos nuestros objetos View
    ListView contactoslstw;

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

        //inicializamos nuestro objeto contactoslstw
        this.contactoslstw = (ListView)
findViewById(R.id.ListaContactosLst);

        //creamos una lista temporal que es un arreglo de cadenas

        String[] milista = {
            "Elemento 1",
            "Elemento 2",
        }
}
```

```
        "Elemento 3",
        "Elemento 4",
        "Elemento 5",
        "Elemento 6",
        "Elemento 7",
        "Elemento 8",
        "Elemento 9",
        "Elemento 10"
    };

    //mandamos a inflar el listview
    this.contactoslstw.setAdapter(new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,mili
sta));
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
}
}
```

Probamos la aplicación en el emulador y vemos que nuestra lista contiene los 10 elementos y esta puede deslizarse

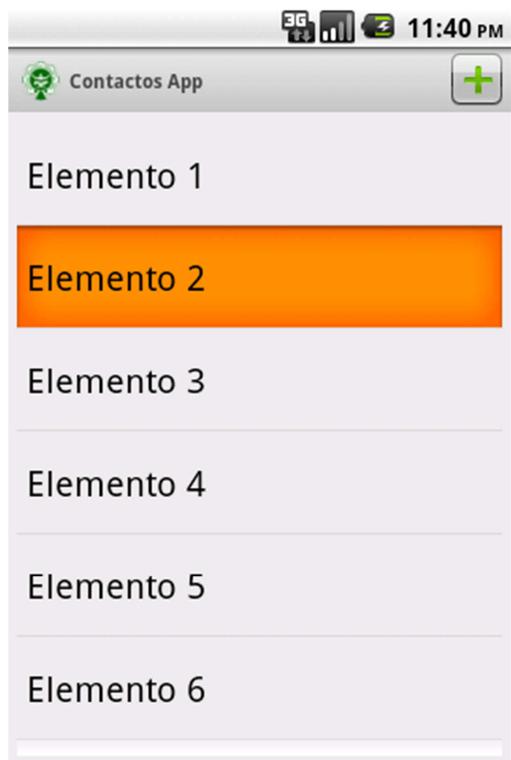


Ilustración 27.- Resultado obtenido con un arreglo de cadenas

Creando un Adaptador

Adaptar una colección de objetos de la clase Contacto (modelo) a la lista.

Adaptar una colección de objetos a un listview requiere de un poco de más trabajo, ya que Android incluye un adaptador por default que funciona con un arreglo de strings, tal y como se mostró en el ejemplo en la línea

```
this.contactoslstw.setAdapter(new  
ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,mili  
sta));
```

La clase ArrayAdapter solicita la sintaxis

```
android.widget.ArrayAdapter.ArrayAdapter<String>(Context context, int textViewResourceId, String[] objects)
```

Por lo tanto no será posible utilizarla para trabajar con la colección de objetos, vemos también que uno de los parámetros es un recurso Layout el cual da forma a cada uno de los elementos del Listview.

Crear un layout para dar forma a cada elemento de la lista

Si lo que queremos es mostrar el nombre completo y algún otro dato de cada contacto en la lista, entonces creamos un layout para representarlo.

Creamos un nuevo layout llamado list_item_contacto.xml

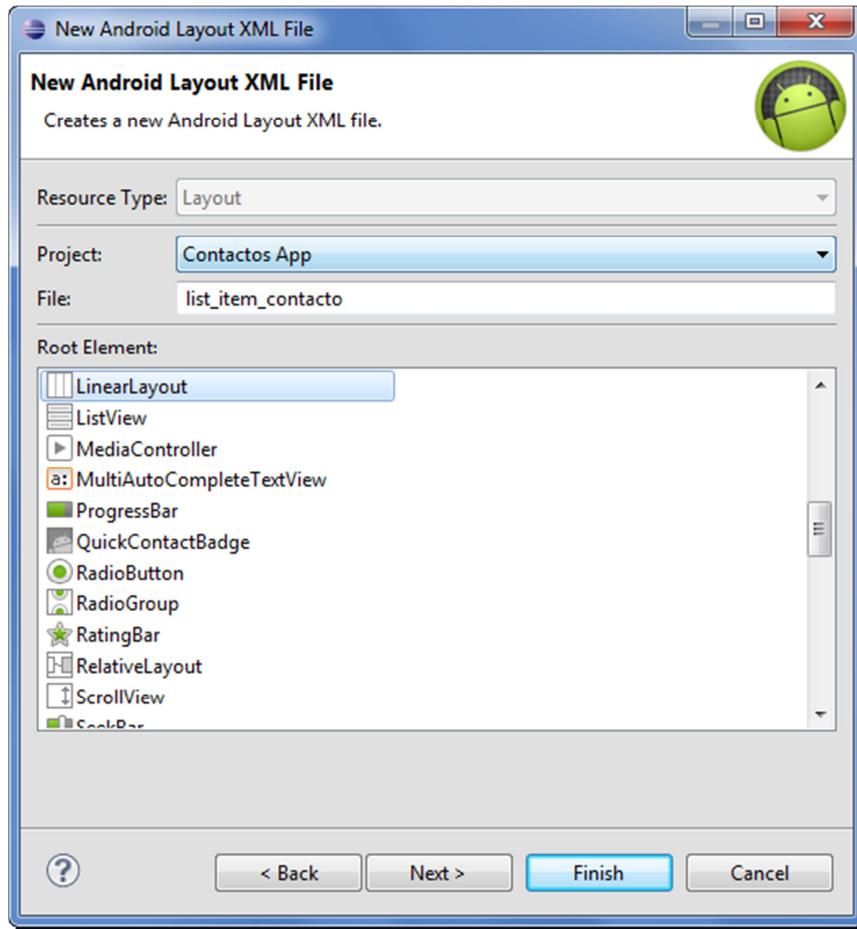


Ilustración 28.- Agregar nuevo layout

Agregamos 2 TextView al nuevo layout

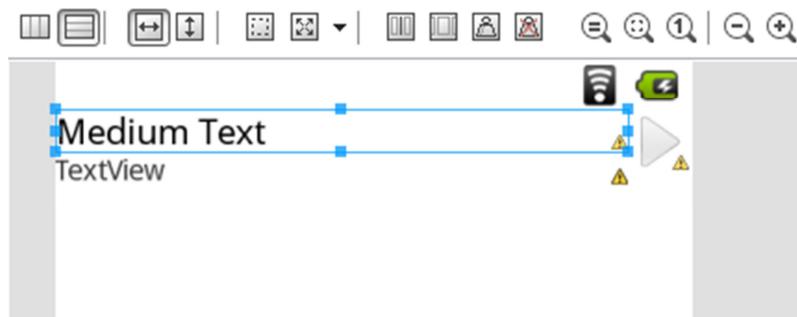


Ilustración 29.- Diseño del layout para representar a cada elemento de la lista

La estructura de este layout queda de esta forma

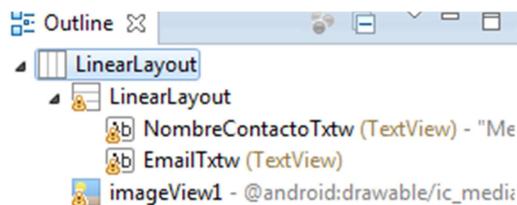


Ilustración 30.- Estructura del layout list_item_contacto

El adaptador para el ListView

Agregamos una clase de nombre ContactoAdapter al paquete de los adapters, esta clase deberá heredar la subclase ArrayAdapter para el tipo Contacto.

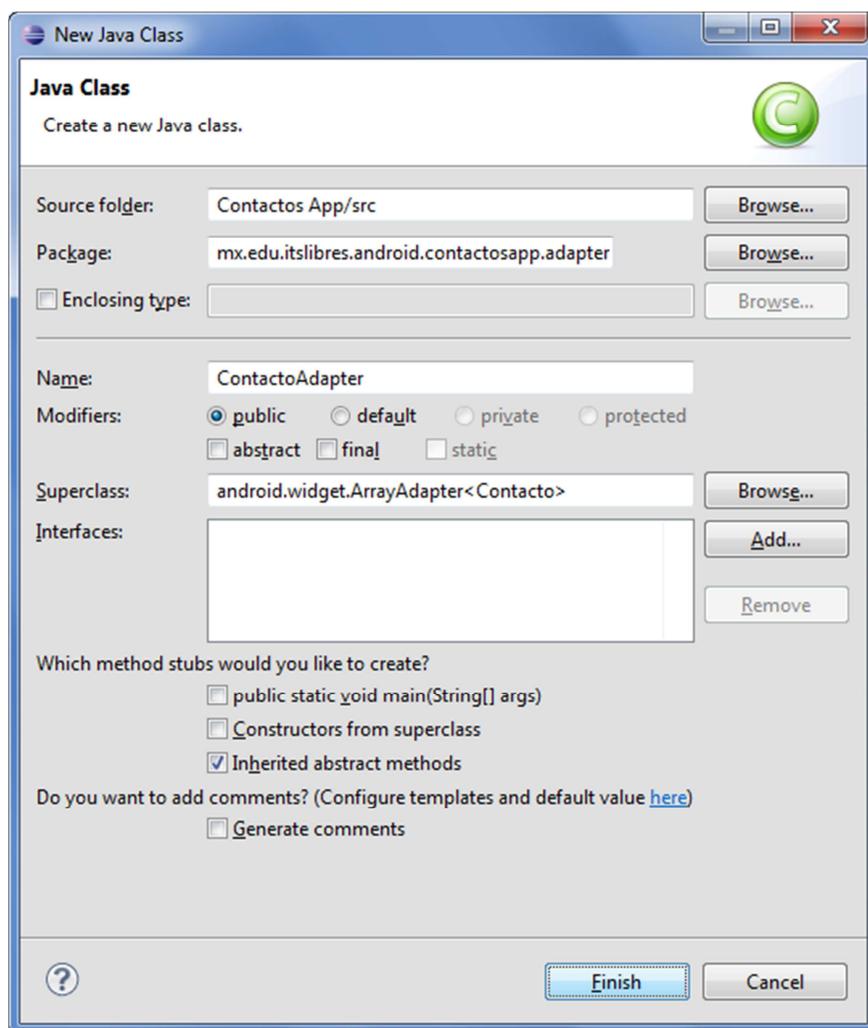


Ilustración 31.- Agregar nueva clase que funcionará como adaptador

Nuestra clase adaptador queda de esta forma:

```
package mx.edu.itslibres.android.contactosapp.adapters;
```

```
import java.util.List;

import mx.edu.itslibres.android.contactosapp.R;
import mx.edu.itslibres.android.contactosapp.models.Contacto;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

public class ContactoAdapter extends ArrayAdapter<Contacto> {

    Context context;
    List<Contacto> contactos;
    //Agregamos el constructor que pida uno de los parametros
    como List
    //para no trabajar directo con los arrays[]
    public ContactoAdapter(Context context, List<Contacto>
contactos) {

        //inicializamos el adaptador, donde definimos el layout
        a utilizar
        //para representar cada uno de los elementos de la lista
        super(context, R.layout.list_item_contacto, contactos);
        this.context = context;
        this.contactos = contactos;
    }

    //implementamos el metodo getView() de la super clase
    //que servira para dar forma a cada uno de los elementos

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

        //Creamos el objeto que inflara cada elemento
        LayoutInflater inflater =
(LayoutInflater)context.getSystemService(Context.LAYOUT_INFLATER_S
ERVICE);

        //Creamos el elementos de la lista
        View item =
inflater.inflate(R.layout.list_item_contacto, null);
    }
}
```

```
//mandamos a asignar los datos en cada textview del elemento
        TextView nombre = (TextView)
item.findViewById(R.id.NombreContactoTxtw);
        nombre.setText(contactos.get(position).getNombre());

        TextView email = (TextView)
item.findViewById(R.id.EmailTxtw);
        email.setText(contactos.get(position).getEmail());
        return item;
    }
}
```

Probemos nuestro adaptador

Modificaremos el MainActivity, donde quitaremos el arreglo de cadenas y la instrucción que manda a inflar el listview, ahora utilizaremos nuestro adaptador para mandar a llenar el listview pero con una lista de objetos Contacto como prueba utilizando el nuevo adaptador.

```
package mx.edu.itslibres.android.contactosapp.activities;

import java.util.ArrayList;
import java.util.List;
import mx.edu.itslibres.android.contactosapp.R;
import
mx.edu.itslibres.android.contactosapp.adapters.ContactoAdapter;
import mx.edu.itslibres.android.contactosapp.models.Contacto;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ListView;

public class MainActivity extends Activity {

    //declaramos nuestros objetos View
    ListView contactoslstw;

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

        //inicializamos nuestro objeto contactoslstw
        this.contactoslstw = (ListView)
findViewById(R.id.ListaContactosLst);
    }
}
```

```
//Creamos una lista temporal de objetos de la clase
Contacto, para
//mostrarlos como prueba a la interfaz

List<Contacto> contactos = new ArrayList();

//llenaremos la lista temporal con 8 objetos Contacto
for(int i=1; i<=8; i++)
{
    Contacto contacto = new Contacto();
    contacto.setId(i);
    contacto.setNombre("Contacto ");
    contacto.setApellidos(" x" + i);
    contacto.setEmail("email"+i+"@ejemplo.com");
    contacto.setTel(i""+i""+i);

    //agregamos el objeto a la lista de contactos
    contactos.add(contacto);
}

//llamamos al nuevo adaptador con la lista de objetos
this.contactoslw.setAdapter(new ContactoAdapter(this,
contactos));

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
}
}
```

El resultado obtenido es el siguiente

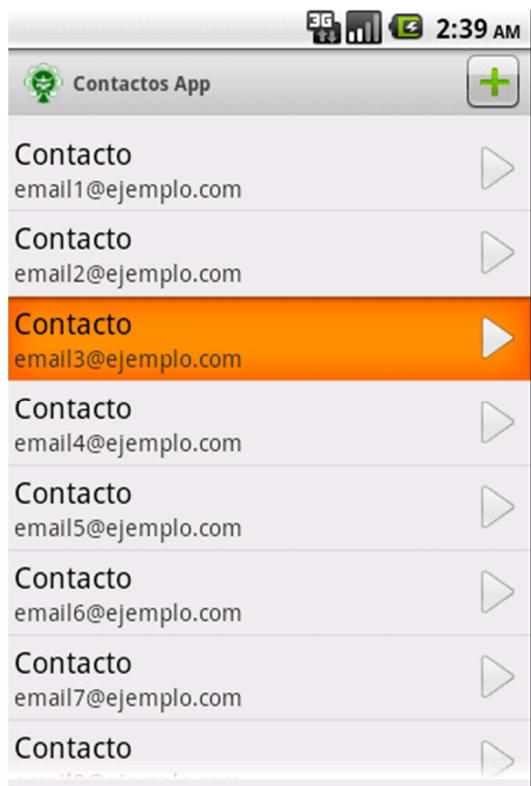


Ilustración 32.- Resultado del adaptador para una lista de objetos

Identificador de Elementos (ID de contacto)

Nota: Para poder identificar cada uno de los elementos de la lista, recomendable agregar un textview oculto donde agregaremos el identificador del registro, agregarlo dentro del list_item_contacto y modificar el adaptador para que asigne el id a este último textview.

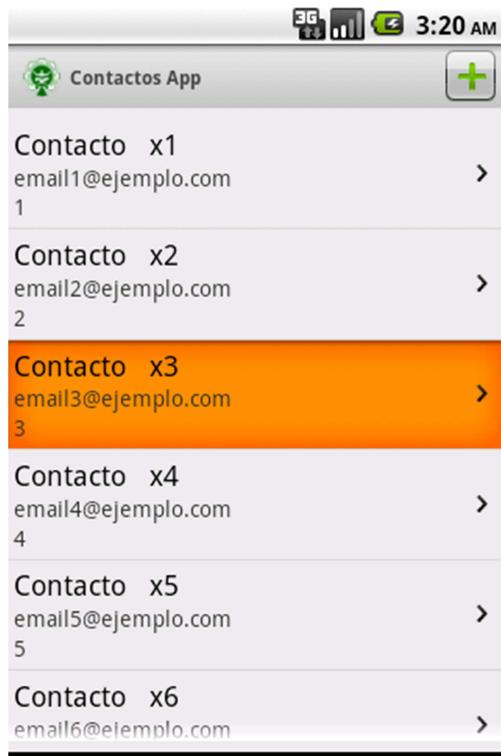


Ilustración 33.- Ahora se visualiza el ID

Diseño de layout y creación de Activities: Registrar y Detalles

Layout para agregar nuevo Contacto

Agregamos un nuevo layout donde agregaremos el formulario para editar o registrar un nuevo contacto, lo llamaremos contacto_form.xml.

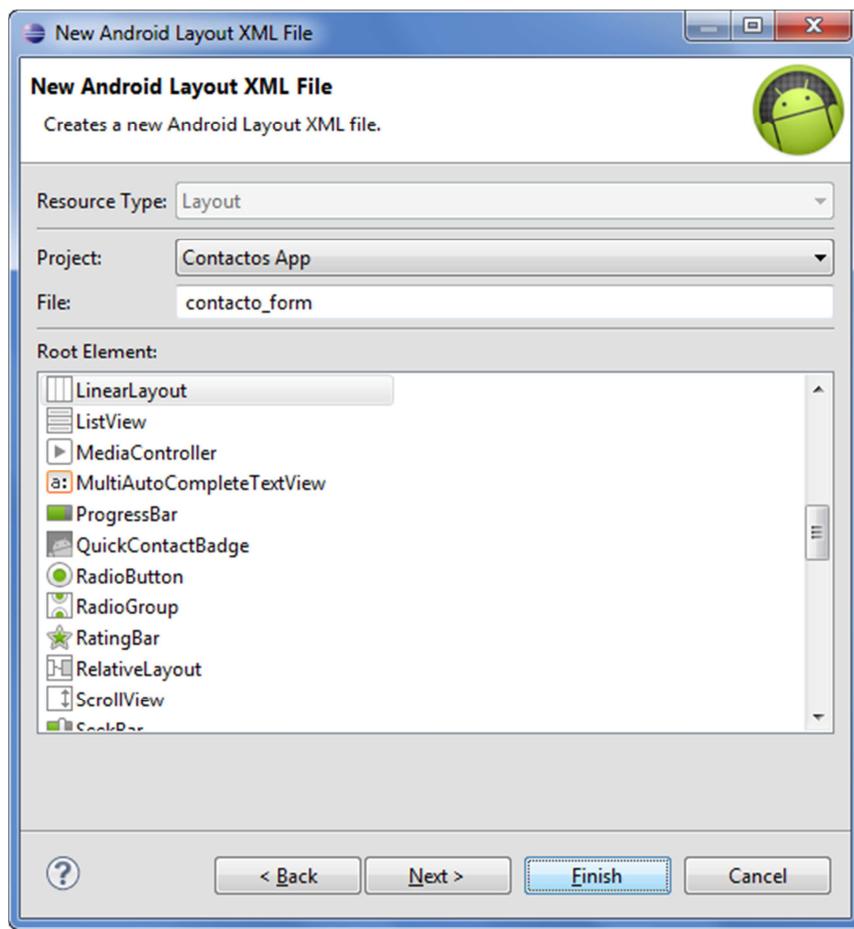


Ilustración 34.- Agregar nuevo layout

Pero antes creamos un Nuevo layout para utilizarlo como encabezado, a este agregaremos 2 botones, uno para cancelar la acción y otro para aceptar los datos y guardarlos (le llamamos `encabezado_form.xml`).

Diseño del encabezado para el formulario



Ilustración 35.- Diseño del encabezado del formulario

Agregamos el nuevo encabezado al layout que utilizaremos como formulario

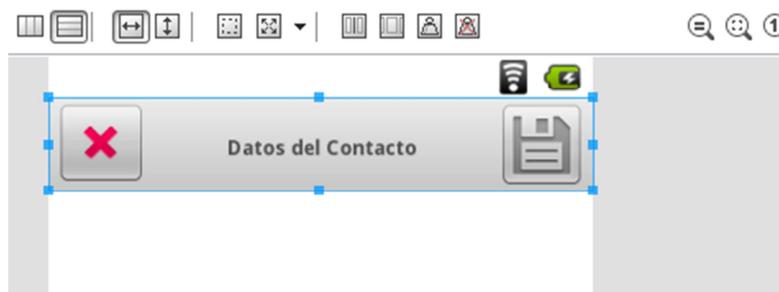


Ilustración 36.- Uso del nuevo encabezado

Diseñamos nuestro formulario

Debajo del encabezado, agregamos un ScrollView para poder hacer que los elementos se deslizen si estos no caben completamente en el alto del layout, al linearlayout principal le agregamos el fondo de aplicación.

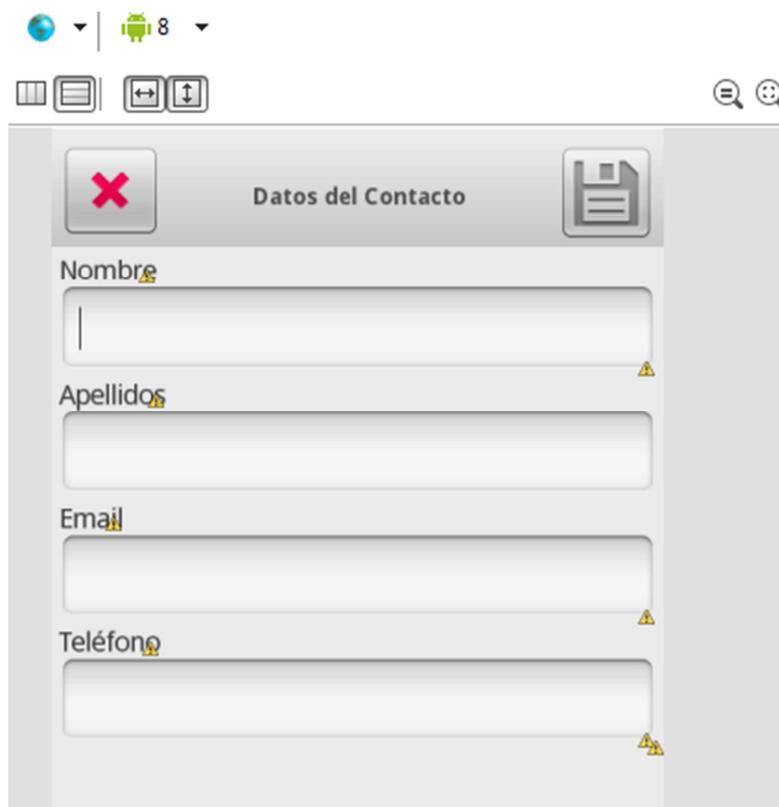


Ilustración 37.- Diseño del formulario

Dándole un toque a los TextView y EditText

Creamos un nuevo estilo para los textview del form al que llamamos **estilolabel**

```
<style name="estilolabel">
    <item name="android:textColor">#333333</item>
    <item name="android:textSize">13sp</item>
    <item name="android:paddingBottom">5dp</item>
```

```
<item name="android:paddingLeft">3dp</item>
<item name="android:paddingTop">4dp</item>
</style>
```

Lo aplicamos a la propiedad Style de todos los textview del formulario el cual ha quedado de esta manera.

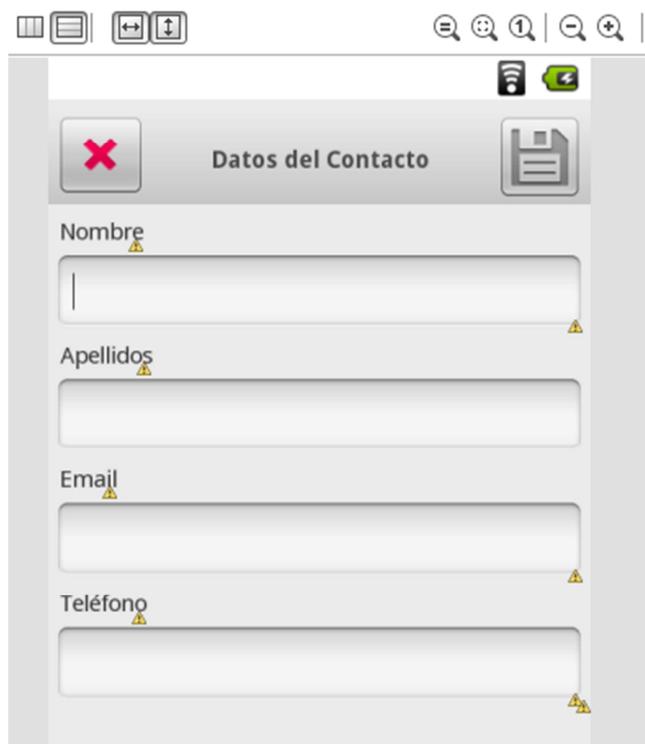


Ilustración 38.- Aplicando estilos a los elementos

Layout para detalles de Contacto

Diseñamos también el layout donde se mostrarán los datos del usuario (como vista de perfil) en el cual utilizaremos el encabezado de aplicación (**detalles_contacto.xml**) donde es opcional crear estilos para modificar la apariencia de los elementos.



Ilustración 39.- Diseño de la interfaz de detalles

Asignar los Id correspondientes a cada elemento que mostrará la información y los botones.

Crear Activity AgregarContactoActivity

Ya creados los layouts, necesitamos ahora crear los Activitys en los cuales se utilizarán los nuevos layouts.

Agregamos una nueva activity (clase) llamada **AgregarContactoActivity** que hereda la superclase Activity.

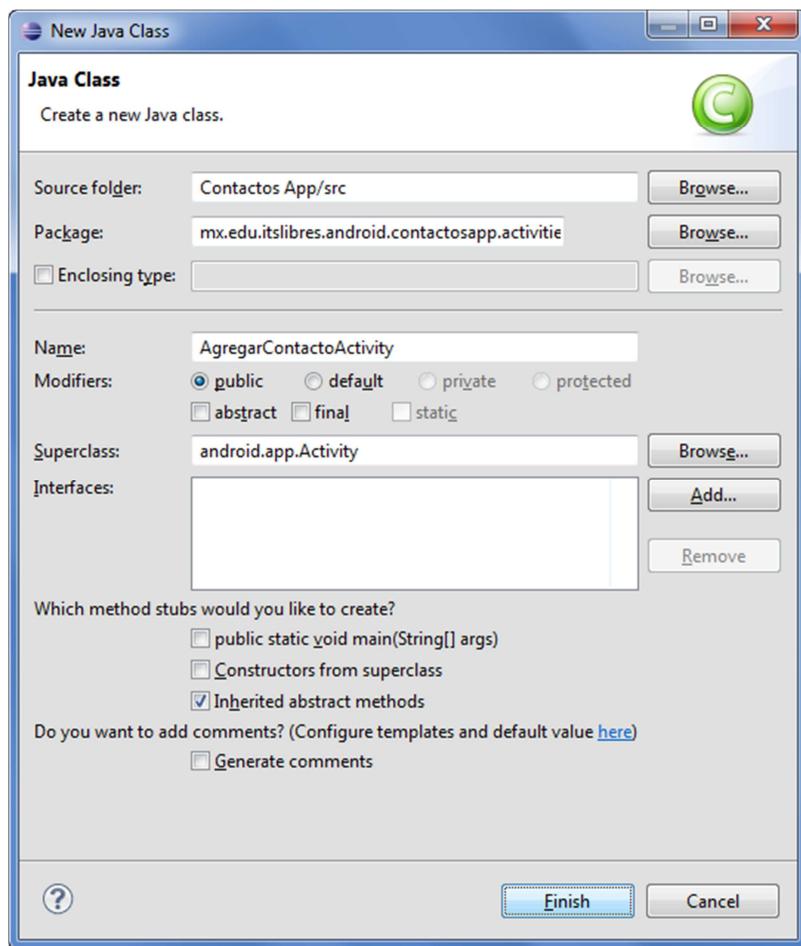


Ilustración 40.- Agregar nuevo activity

En esta, asignamos el layout del formulario.

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import android.app.Activity;
import android.os.Bundle;

//agregamos el constructor
public class AgregarContactoActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //asignamos el layout contacto_form
        setContentView(R.layout.contacto_form);
    }
}
```

Crear Activity DetallesContactoActivity

Agregamos una nueva activity **DetallesContactoActivity**

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import android.app.Activity;
import android.os.Bundle;

public class DetallesContactoActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //asignamos el layout contacto_form
        setContentView(R.layout.detalles_contacto);
    }
}
```

Controlar eventos de los botones

Implementemos el evento click para el botón Agregar que se encuentra en el layout principal.

En MainActivity, declaramos el objeto ImageButton

```
ImageButton AgregarContactoBtn;
```

Dentro de onCreate() lo inicializamos

```
this.AgregarContactoBtn = (ImageButton)
findViewById(R.id.AgregarCImgb);
```

Agregamos el código para controlar el evento Click para este botón (dentro de onCreate)

```
//implementamos el evento clic para AgregarContactoBtn
this.AgregarContactoBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // hacer que abra el nuevo layout
        Intent agregarcontacto = new
Intent(MainActivity.this, AgregarContactoActivity.class);
        startActivity(agregarcontacto);
    }
});
```

```
    }  
});
```

Registrar los Activities

Para que un Activity funcione, es necesario agregarlo a AndroidManifest.xml, de lo contrario se tendrá el error:

The screenshot shows a logcat output in a terminal window. The error message is as follows:

```
FATAL EXCEPTION: main  
    android.content.ActivityNotFoundException: Unable to find explicit  
    activity class {mx.edu.itslibres.android.contactosapp/mx.edu.itslib  
    res.android.contactosapp.activities.AgregarContactoActivity}; have  
    you declared this activity in your AndroidManifest.xml?  
        at android.app.Instrumentation.checkStartActivityResult(Instrumen  
        tation.java:1404)  
        at android.app.Instrumentation.execStartActivity(Instrumentation.j  
        ava:1378)  
        at android.app.Activity.startActivityForResult(Activity.java:2817)  
        at android.app.Activity.startActivity(Activity.java:2923)  
        at mx.edu.itslibres.android.contactosapp.activities.MainActivity$1  
        .onClick(MainActivity.java:79)
```

Ilustración 41.- Error cuando no se registra un Activity

En el archivo de manifiesto, agregamos los 2 activities nuevos.

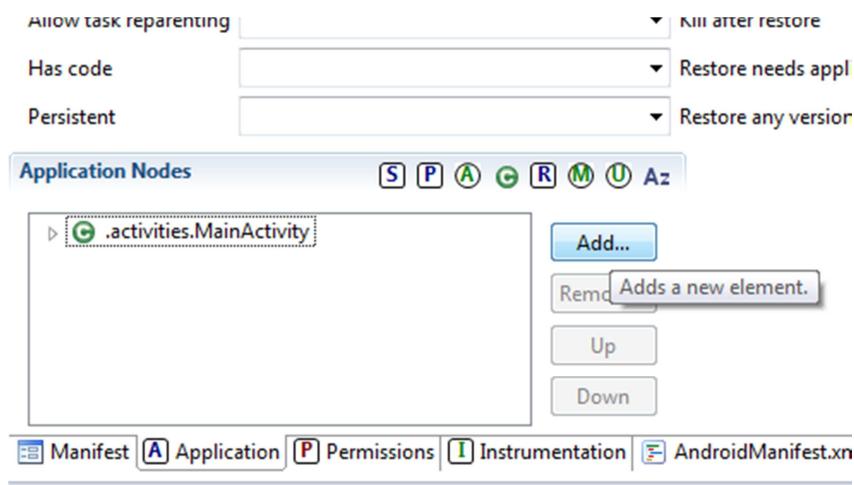


Ilustración 42.- Registrando un activity

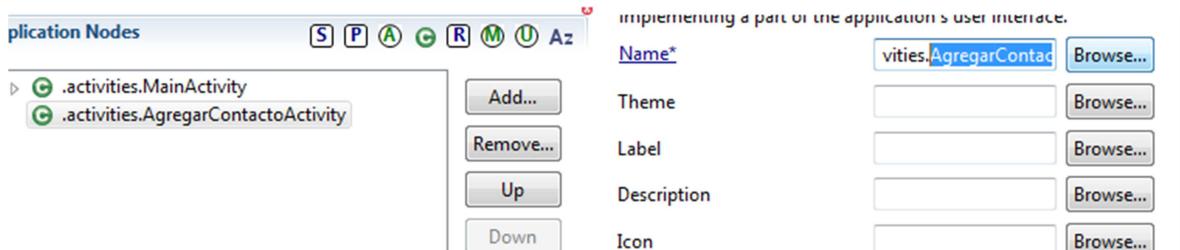


Ilustración 43.- Activity registrado

Probando el resultado logrado hasta el momento

Al correr la aplicación, y presionar el botón Agregar Contacto, nos llevará a la nueva activity



Ilustración 44.- Vista de funcionalidad

Programar los botones en AgregarContactoActivity

En AgregarContactoActivity, implementar el evento Click para el botón Cancelar, y dentro de este agregar un **finish()** para terminar la activity y volver a la anterior.

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;
```

```
//agregamos el constructor
public class AgregarContactoActivity extends Activity {

    ImageButton CancelarBtn;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //asignamos el layout contacto_form
        setContentView(R.layout.contacto_form);

        this.CancelarBtn=                               (ImageButton)
findViewById(R.id.CancelarBtn);

        this.CancelarBtn.setOnClickListener(new
View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                //regresar a la anterior sin hacer nada
                finish();
            }
        });
    }
}
```

También programar el **botón Guardar**, por ahora solo emitirá un mensaje diciendo que guardo los datos y terminar.

```
//al presionar el boton Guardar
this.GuardarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {

        //emitir un mensaje
        Toast.makeText(getApplicationContext(), "Los datos
del contacto se guardarán", Toast.LENGTH_SHORT).show();
        finish();

    }
});
```

Ahora al presionar el botón Guardar, tendremos el resultado



Ilustración 45.- Al presionar Guardar

Controlar el evento al seleccionar un elemento de la lista de Contactos

Volvemos al MainActivity donde controlaremos el evento click sobre los elementos el ListView (antes declararlo e inicializarlo).

```
this.ContactosListView.setOnItemClickListener(new  
AdapterView.OnItemClickListener() {  
  
    @Override  
    public void onItemClick(AdapterView<?> parent,  
View view, int position,  
    long id) {  
        // TODO Auto-generated method stub  
  
        //aqui se recuperaran los datos del elemento  
  
        //pasamos a mostrar el activity de los  
detalles
```

```
        Intent detalles = new Intent(MainActivity.this, DetallesContactoActivity.class);
        startActivity(detalles);

    }
});
```

Al probar la aplicación y seleccionar uno de los elementos de la vista nos llevará al nuevo activity



Ilustración 46.- Vista del activity en función

Pasar datos entre Activities.

Donde se mencionó que se recuperarán los datos del contacto, solo obtendremos el id que es visible por ahora en la lista (después lo ocultaremos).

Ya recuperado el id, lo enviaremos al nuevo activity.

Por lo tanto el código del controlador de evento click del listview quedará así:

```
this.ContactosListView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent,
View view, int position,
```

```

        long id) {
    // TODO Auto-generated method stub

    //aqui se recuperaran los datos del elemento
    //recuperamos solo el ID
    TextView idcontacto =
(TextView)view.findViewById(R.id.IdContactoTxtw);

    //Creamos el objeto que ayudara a llevar los
    datos
    Bundle datosextras = new Bundle();
    datosextras.putInt("Id",
Integer.valueOf(idcontacto.getText().toString()));

    //pasamos a mostrar el activity de los
    detalles
    Intent detalles = new
Intent(MainActivity.this, DetallesContactoActivity.class);
    //antes de llamar al intent, agregamos los
    extras
    detalles.putExtras(datosextras);

    startActivity(detalles);

}
});

```

En el activity **DetallesContactoActivity**, recibimos el **id** del contacto. El código completo de este activity es:

```

package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class DetallesContactoActivity extends Activity {

    //inicializar los elementos de la vista
    TextView NombreLbl;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //asignamos el layout contacto_form

```

```
setContentView(R.layout.detalles_contacto);

this.NombreLbl = (TextView) findViewById(R.id.NombreLbl);

//Recuperar los datos recibidos del activity que invoco
//si se tienen datos entonces procesarlos
if(getIntent().getExtras() != null)
{
    Bundle extras = getIntent().getExtras();
    int idcontacto = extras.getInt("Id");

    //Cuando se tenga conexion al webservice, se
recuperará el objeto y se
        //mostrarán en la vista

    //por ahora mostramos solo el id
    this.NombreLbl.setText("Nombre del contacto " +
idcontacto);
}
}
```

Entonces al probar el ejemplo y seleccionar un elemento de la lista, obtenemos:

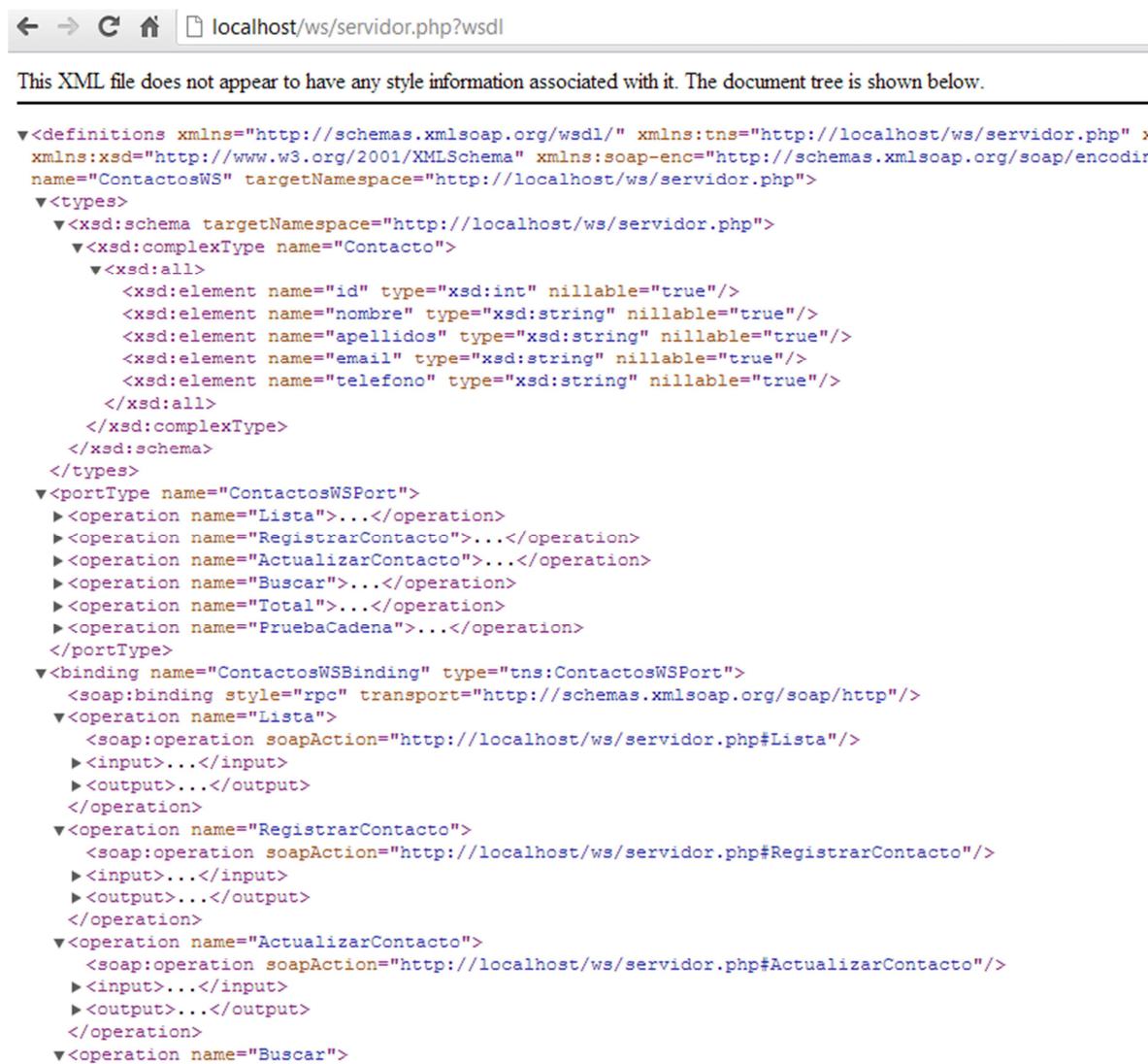


Ilustración 47.- Activity que ha recibido datos

En esta activity programar también el botón agregar.

Conectar la aplicación al Web Service

Ya listo el diseño, ahora implementaremos las partes para permitir a nuestra aplicación conectarse a un servicio web provisto por un servidor web.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://localhost/ws/servidor.php" x
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encodin
  name="ContactosWS" targetNamespace="http://localhost/ws/servidor.php">
  <types>
    <xsd:schema targetNamespace="http://localhost/ws/servidor.php">
      <xsd:complexType name="Contacto">
        <xsd:all>
          <xsd:element name="id" type="xsd:int" nillable="true"/>
          <xsd:element name="nombre" type="xsd:string" nillable="true"/>
          <xsd:element name="apellidos" type="xsd:string" nillable="true"/>
          <xsd:element name="email" type="xsd:string" nillable="true"/>
          <xsd:element name="telefono" type="xsd:string" nillable="true"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <portType name="ContactosWSPort">
    <operation name="Lista">...</operation>
    <operation name="RegistrarContacto">...</operation>
    <operation name="ActualizarContacto">...</operation>
    <operation name="Buscar">...</operation>
    <operation name="Total">...</operation>
    <operation name="PruebaCadena">...</operation>
  </portType>
  <binding name="ContactosWSBinding" type="tns:ContactosWSPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Lista">
      <soap:operation soapAction="http://localhost/ws/servidor.php#Lista"/>
      <input>...</input>
      <output>...</output>
    </operation>
    <operation name="RegistrarContacto">
      <soap:operation soapAction="http://localhost/ws/servidor.php#RegistrarContacto"/>
      <input>...</input>
      <output>...</output>
    </operation>
    <operation name="ActualizarContacto">
      <soap:operation soapAction="http://localhost/ws/servidor.php#ActualizarContacto"/>
      <input>...</input>
      <output>...</output>
    </operation>
    <operation name="Buscar">
```

Ilustración 48.- Servicio web al que intentamos acceder

La librería ksoap2-android

Utilizaremos la librería ksoap2-android [The ksoap2-android project provides a lightweight and efficient SOAP client library for the Android platform]

<http://code.google.com/p/ksoap2-android/>

Añadir la librería ksoap2-android

Arrastrar el archivo .jar a la carpeta **libs** del proyecto

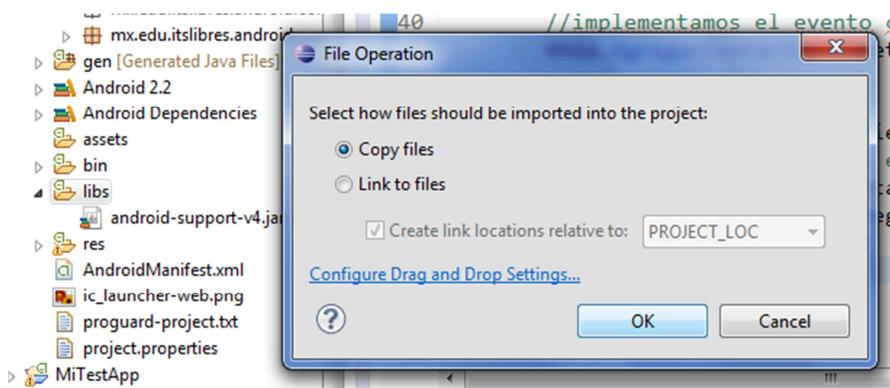


Ilustración 49.- Agregando librerías externas

Automáticamente se agrega a Android Dependencies

Creando las clases necesarias para el acceso a datos.

En el paquete dataaccess, agregamos una nueva clase llamada AccesoServiciosWeb. Esta clase será genérica, podrá ser utilizada para acceder a cualquier método de cualquier servicio web.

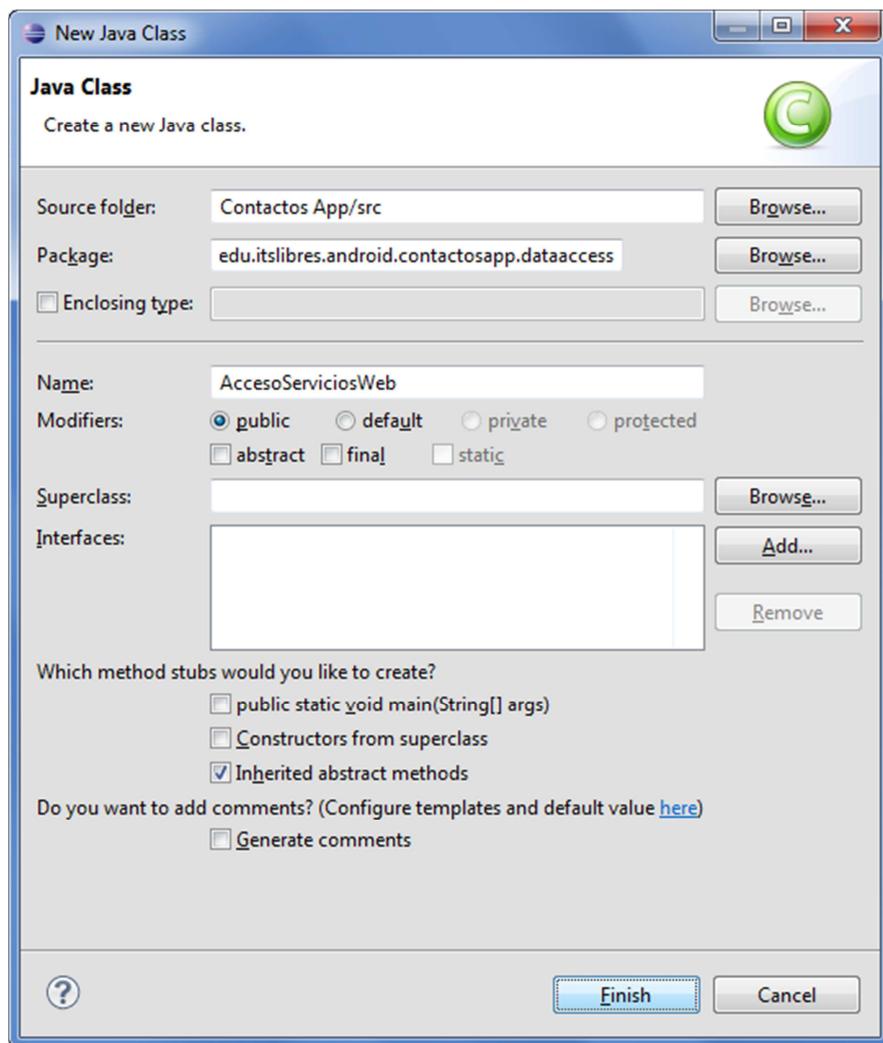


Ilustración 50.- Agregando clase para acceder a los WS

La clase quedará como:

```
package mx.edu.itslibres.android.contactosapp.dataaccess;

import java.util.Map;
import java.util.Map.Entry;

import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;

import android.util.Log;

public final class AccesoServiciosWeb {
```

```

//Método que invocará al WS y devolverá respuesta como objeto
soap
public SoapObject AccesoServicio(String EspacioNombres,
String Metodo, String URL, Map<String, String> parametros)
{
    //variables necesarias para la conexión y procesamiento
    de datos
    String soap_accion = EspacioNombres +"#" +Metodo;
    SoapObject request;
    SoapSerializationEnvelope envelope;
    request = new SoapObject(EspacioNombres, Metodo);

    Log.d("Llamando a servicio:", soap_accion);
    Log.d("Url Servicio: ", URL);

    //antes de iniciar la llamada, agregar los parametros
    (solo si se especificaron)
    //estos parametros corresponderán a los solicitados por
    el método que sea invocado
    if(parametros != null){
        for (Entry<String, String> parametro : parametros.entrySet()) {
            request.addProperty(parametro.getKey(),
parametro.getValue());
            Log.d("Parámetro enviado>", parametro.getKey() +
": valor=" + parametro.getValue());
        }
    }

    envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER12);
    envelope.dotNet = false; // no es un servicio .Net
    envelope.setOutputSoapObject(request);

    HttpTransportSE transporte = new HttpTransportSE(URL);

    try {
        transporte.call(soap_accion, envelope);
        Log.d("Respuesta recibida> ", envelope.bodyIn.toString());

        SoapObject result = (SoapObject) envelope.bodyIn;

        //devolvemos el objeto SOAP
        return result;
    } catch (Exception e) {
}

```

```
        e.printStackTrace();
        return null;
    }
}
```

Agregamos ahora una nueva clase a **dataaccess** que será encargada de acceder a todos los métodos de nuestro servicio web.

La clase la llamaremos **AccesoContactosWS**:

```
package mx.edu.itslibres.android.contactosapp.dataaccess;

import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Vector;

import org.ksoap2.serialization.SoapObject;

import mx.edu.itslibres.android.contactosapp.models.Contacto;

public final class AccesoContactosWS {
    //creamos una instancia de nuestra clase para acceder a los
    WS
    AccesoServiciosWeb acesoservicio = new AccesoServiciosWeb();

    //nuestra variable que almacene la url del WS
    private String URL_WebServices =
    "http://192.168.1.227/ws/servidor.php";
    //nombre del espacio de nombres del web service (es la misma
    url)
    private String EspacioNombres =
    "http://192.168.1.227/ws/servidor.php";

    //metodo que devuelva la lista de contactos obtenida desde el
    WS
    @SuppressWarnings({ "unchecked", "rawtypes" })
    public final List<Contacto> ListaContactos(){
        try {
            //acceder al servicio
```

```
        SoapObject             resultado      =
acesoservicio.AccesoServicio(EspacioNombres,           "Lista",
URL_WebServices, null);

        //colección que almacenara la lista de contactos
        //la inicializamos como null para devolverla así
en caso
        //de que no se obtenga ningun resultado
List<Contacto> listac = null;

        if(resultado.getProperty(1).getClass() == 
java.util.Vector.class)
{
    //obtenemos la lista de contactos desde el WS
y la convertimos a List<Contacto>
    listac = new
ConvertContacto().ListaContactos(Collections.List((Enumeration<Soa
pObject>)((Vector)resultado.getProperty(1)).elements()));
}

        //devolver la lista de contactos ya procesada
return listac;

}catch (Exception e) {
    //e.printStackTrace();
    return null;
}
}

//metodo que obtenga los datos de un contacto
public final Contacto DatosContacto(int idContacto){
try {
    Contacto contacto = null;
    //crear los parametros solicitados por el metodo a
llamar
    Map<String,     String>     parametros      =      new
HashMap<String, String>();
    parametros.put("idContacto",
String.valueOf(idContacto));

    //obtenemos el objeto soap que devolvio el WS
    //aqui si le pasamos los parametros
    SoapObject             resultado      =
acesoservicio.AccesoServicio(EspacioNombres,           "DatosContacto",
URL_WebServices, parametros);
```

```
//procesamos el objeto
if(resultado.getProperty(1).getClass() == SoapObject.class)
{
    contacto = new
ConvertContacto().UnicoContacto((SoapObject)resultado.getProperty(1));
}

//devolvemos el contacto
return contacto;

}catch (Exception e) {
    return null;
}
}

//metodo que llama al registro de un nuevo contacto
public final Contacto RegistrarContacto(Contacto contacto){
    try {
        //crear los parametros solicitados por el metodo a
        llamar
        //aqui serán todos los datos de un contacto
        Map<String, String> parametros = new
        HashMap<String, String>();
        parametros.put("nombre", contacto.getNombre());
        parametros.put("apellidos", contacto.getApellidos());
        parametros.put("email", contacto.getEmail());
        parametros.put("telefono", contacto.getTelefono());
        //obtenemos el objeto soap que devolvio el WS
        //aqui si le pasamos los parametros
        SoapObject resultado =
acesoservicio.AccesoServicio(EspacioNombres, "RegistrarContacto",
URL_WebServices, parametros);

        //procesamos el objeto
        if(resultado.getProperty(1).getClass() == SoapObject.class)
{
            contacto = new
ConvertContacto().UnicoContacto((SoapObject)resultado.getProperty(1));
}
}

//devolvemos el contacto
```

```
        return contacto;

    }catch (Exception e) {
        return null;
    }
}

//metodo que llama a la actualizacion de los datos de un
nuevo contacto
public final boolean ActualizarContacto(Contacto
contacto){
    try {
        //crear los parametros solicitados por el
        metodo a llamar
        //aqui seran todos los datos de un contacto
        Map<String, String> parametros = new
HashMap<String, String>();
        parametros.put("id",
String.valueOf(contacto.getId()));
        parametros.put("nombre",
contacto.getNombre());
        parametros.put("apellidos",
contacto.getApellidos());
        parametros.put("email", contacto.getEmail());
        parametros.put("telefono",
contacto.getTelefono());
        //obtenemos el objeto soap que devolvio el WS
        //aqui si le pasamos los parametros
        SoapObject resultado =
acesoservicio.AccesoServicio(EspacioNombres, "ActualizarContacto",
URL_WebServices, parametros);

        //devolvemos el resultado
        return
Boolean.valueOf(String.valueOf(resultado.getProperty(1)));
    }catch (Exception e) {
        return false;
    }
}

//metodo que llama la busqueda de contactos
@SuppressWarnings({ "unchecked", "rawtypes" })
public final List<Contacto> BuscarContactos(String q){
    try {
```

Dentro de esta clase se está invocando a otra llamada **ConvertContacto**, la cual su código es:

```
package mx.edu.itslibres.android.contactosapp.dataaccess;

import java.util.ArrayList;
import java.util.List;
```

```
import mx.edu.itslibres.android.contactosapp.models.Contacto;

import org.ksoap2.serialization.SoapObject;

/*
 * Clase que convierte un vector de objetos SOAP a list<Contacto>
 * Tambien convierte un solo objeto SOAP a objeto Contacto
 */
public final class ConvertContacto {
    public List<Contacto> ListaContactos(List<SoapObject> lista){

        List<Contacto> contactos = new ArrayList<Contacto>();
        for(SoapObject item : lista)
        {
            /*
            Contacto contacto = new Contacto();

            contacto.setId(Integer.parseInt(item.getProperty("id").toString()));

            contacto.setNombre(item.getProperty("nombre").toString());
            contacto.setApellidos(item.getProperty("apellidos").toString());
            contacto.setEmail(item.getProperty("email").toString());
            contacto.setTelefono(item.getProperty("telefono").toString());
            */

            //llamamos al metodo que procesa un solo objeto
            //y lo agregamos a la lista de contactos
            contactos.add(UnicoContacto(item));
        }

        return contactos;
    }

    public Contacto UnicoContacto(SoapObject item){

        Contacto contacto = new Contacto();
```

```
    contacto.setId(Integer.parseInt(item.getProperty("id").toString()));

    contacto.setNombre(item.getProperty("nombre").toString());

    contacto.setApellidos(item.getProperty("apellidos").toString());
    contacto.setEmail(item.getProperty("email").toString());

    contacto.setTelefono(item.getProperty("telefono").toString())
;

    return contacto;
}
}
```

Hasta aquí nuestras clases para acceder al WS ya están listas.

Agregar permisos de red a la aplicación

Para que la aplicación pueda conectarse a través de la red, es necesario concederle el permiso de Internet.

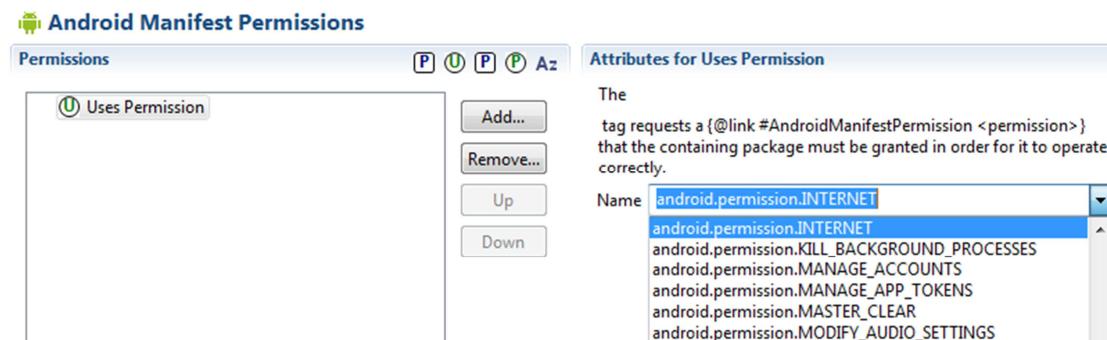


Ilustración 51.- Agregando permisos a la app

Manipulando los datos a través del Web Service

Ahora reemplazaremos la colección de objetos temporal en el MainActivity, invocaremos la llamada del WS y el resultado lo mandaremos a mostrar en el layout.

Agregamos un objeto de la clase AccesoContactosWS como miembro de MainActivity.

```
AccesoContactosWS accesoContactosWS;
```

Donde llamábamos al adaptador, ahora lo reemplazamos por:

```
//llamamos al nuevo adaptador con la lista de objetos  
//this.contactoslstw.setAdapter(new ContactoAdapter(this,  
contactos));  
  
try{  
    accesoContactosWS = new AccesoContactosWS();  
    this.contactoslstw.setAdapter(new ContactoAdapter(this,  
accesoContactosWS.ListaContactos()));  
}catch(Exception e)  
{  
    Toast.makeText(this, "Ocurrió un error al cargar  
la lista de contactos.", Toast.LENGTH_SHORT).show();  
    //finish();  
}
```

Al correr la aplicación, ya podemos ver la lista de los contactos registrados:

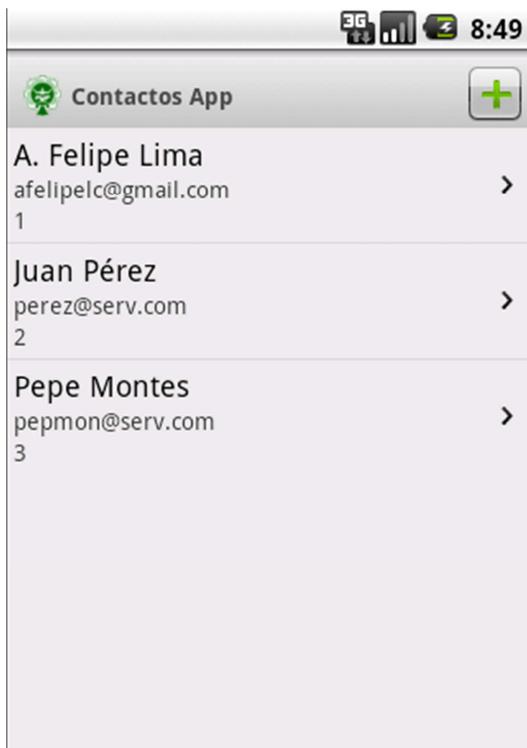


Ilustración 52.- Respuesta del WS

Son los registros de nuestra BD

The screenshot shows a MySQL Workbench interface. At the top, there is a SQL query window with the following code:

```
SELECT *
FROM `contactos`
LIMIT 0 , 30
```

Below the query window is a toolbar with several buttons. Underneath the toolbar, there are search and filter fields. The main area displays a table of contact data with the following columns: id, nombre, apellidos, email, and telefono. The data in the table is:

	id	nombre	apellidos	email	telefono
<input type="checkbox"/>	1	A. Felipe	Lima	afelipelc@gmail.com	2223333
<input type="checkbox"/>	2	Juan	Pérez	perez@serv.com	456476
<input type="checkbox"/>	3	Pepe	Montes	pepmont@serv.com	765432

At the bottom of the table area, there are buttons for 'Marcar todos / Desmarcar todos' (Select all / Deselect all), 'Cambiar' (Change), and a delete icon.

Ilustración 53.- Registros de nuestra BD

Hace falta ahora programar el activity **DetallesContactoActivity** para mostrar los todos datos, este ya recibe el ID, solo hace falta obtener los datos del WS y mandar a mostrarlos en los elementos del layout.

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS
;
import mx.edu.itslibres.android.contactosapp.models.Contacto;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

public class DetallesContactoActivity extends Activity {

    //inicializar los elementos de la vista
    TextView NombreLbl;
    TextView ApellidosLbl;
    TextView EmailLbl;
    TextView TelefonoLbl;

    ImageButton AgregarContactoBtn;
```

```
AccesoContactosWS accesoContactosWS;
Contacto contacto;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //asignamos el layout contacto_form
    setContentView(R.layout.detalles_contacto);

    this.AgregarContactoBtn = (ImageButton)
    findViewById(R.id.AgregarCImgb);
    this.NombreLbl = (TextView) findViewById(R.id.NombreLbl);
    this.ApellidosLbl = (TextView)
    findViewById(R.id.ApellidosLbl);
    this.EmailLbl = (TextView) findViewById(R.id.EmailLbl);
    this.TelefonoLbl =
    (TextView) findViewById(R.id.TelefonoLbl);
    //Recuperar los datos recibidos del activity que invoco
    //si se tienen datos entonces procesarlos
    if(getIntent().getExtras() != null)
    {
        accesoContactosWS = new AccesoContactosWS();

        Bundle extras = getIntent().getExtras();
        int idcontacto = extras.getInt("Id");

        contacto
        accesoContactosWS.DatosContacto(idcontacto);
        if(contacto!=null)
        {
            //Cuando se tenga conexion al webservice, se
            recuperará el objeto y se
            //mostrarán en la vista

            //por ahora mostramos solo el id
            this.NombreLbl.setText(contacto.getNombre());
            this.ApellidosLbl.setText(contacto.getApellidos());
            this.EmailLbl.setText(contacto.getEmail());
            this.TelefonoLbl.setText(contacto.getTelefono());

        }else
            Toast.makeText(getApplicationContext(), "No se pudo
            obtener los datos del contacto", Toast.LENGTH_SHORT).show();
    }
}
```

```
//implementamos el evento clic para AgregarContactoBtn
this.AgregarContactoBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // hacer que abra el nuevo layout
        Intent agregarcontacto = new
Intent(DetallesContactoActivity.this,
AgregarContactoActivity.class);
        startActivity(agregarcontacto);
    }
})
}
```

Al seleccionar un elemento de la lista, obtenemos:



Ilustración 54.- Detalles del contacto

Editar los datos del contacto

Hace falta entonces Agregar un nuevo Activity para editar los datos del contacto, posteriormente programar el botón Editar.

Creamos el activity llamado EditarContactoActivity y lo registramos en el manifiesto.

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS
;
import mx.edu.itslibres.android.contactosapp.models.Contacto;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.Toast;

public class EditarContactoActivity extends Activity {

    AccesoContactosWS accesoContactosWS;
    Contacto contacto;

    EditText NombreTxt;
    EditText ApellidosTxt;
    EditText EmailTxt;
    EditText TelefonoTxt;

    ImageButton CancelarBtn;
    ImageButton GuardarBtn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.contacto_form);

        //inicializar los elementos de la vista
        this.NombreTxt = (EditText)findViewById(R.id.NombreTxt);

        this.ApellidosTxt=(EditText)findViewById(R.id.ApellidosTxt);
        this.EmailTxt = (EditText)findViewById(R.id.EmailTxt);
        this.TeléfonoTxt =
        (EditText)findViewById(R.id.TeléfonoTxt);
```

```

this.CancelarBtn=                               (ImageButton)
findViewById(R.id.CancelarBtn);
    this.GuardarBtn                      =           (ImageButton)
findViewById(R.id.GuardarBtn);

//Recuperar los datos recibidos del activity que invoco
//si se tienen datos entonces procesarlos
if(getIntent().getExtras() != null)
{
    accesoContactosWS = new AccesoContactosWS();

    Bundle extras = getIntent().getExtras();
    int idcontacto = extras.getInt("Id");

    contacto
accesoContactosWS.DatosContacto(idcontacto);
    if(contacto!=null)
    {
        //Cuando se tenga conexion al webservice, se
recuperará el objeto y se
            //mostrarán en la vista

        //por ahora mostramos solo el id
        this.NombreTxt.setText(contacto.getNombre());
        this.ApellidosTxt.setText(contacto.getApellidos());
        this.EmailTxt.setText(contacto.getEmail());
        this.TelefonoTxt.setText(contacto.getTelefono());

    }else
        Toast.makeText(getApplicationContext(), "No se pudo
obtener los datos del contacto", Toast.LENGTH_SHORT).show();
    }

//al presionar el boton cancelar
this.CancelarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        //regresar a la anterior sin hacer nada
        finish();

    }
});
```

```
//al presionar el boton Guardar
this.GuardarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {

        //emitir un mensaje
        Toast.makeText(getApplicationContext(), "Los datos
del contacto se guardarán", Toast.LENGTH_SHORT).show();
        finish();

    }
});}

}
```

Ahora programamos el botón Editar de Detalles de contacto, y pasamos el **id** al activity EditarContacto.

```
this.EditarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {

        //Creamos el objeto que ayudara a llevar los
datos
        Bundle datosextras = new Bundle();
        datosextras.putInt("Id",
Integer.valueOf(contacto.getId()));

        Intent editarcontacto = new
Intent(DetallesContactoActivity.this,
EditarContactoActivity.class);
        editarcontacto.putExtras(datosextras);
        startActivity(editarcontacto);

    }
});
```

Al correr la aplicación y estando en Detalles del contacto presionamos Editar tenemos:



Ilustración 55.- Editar contacto

Ahora hace falta programar el botón Guardar de EditarContactoActivity, para que este invoque al servicio web mandando a guardar el registro.

```
//al presionar el boton Guardar
this.GuardarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {

        //emitir un mensaje
        //Toast.makeText(getApplicationContext(), "Los datos
del contacto se guardarán", Toast.LENGTH_SHORT).show();

        //antes de mandar a guardar, recuperar los
nuevos datos de los Txt

        contacto.setNombre(NombreTxt.getText().toString());

        contacto.setApellidos(ApellidosTxt.getText().toString());
    }
})
```

```
contacto.setEmail(EmailTxt.getText().toString());  
  
contacto.setTelefono(TelefonoTxt.getText().toString());  
  
try{  
  
    if(accesoContactosWS.ActualizarContacto(contacto))  
        Toast.makeText(getApplicationContext(),  
"Los datos del contacto han sido guardados",  
Toast.LENGTH_SHORT).show();  
    else  
        Toast.makeText(getApplicationContext(),  
"Ocurrió un error al actualizar los datos del contacto",  
Toast.LENGTH_SHORT).show();  
  
    finish();  
  
}catch(Exception e){  
    finish();  
}  
});
```

Al correr la aplicación y editar un registro, comprobamos que ya funciona y se regresa a la vista de los datos, pero no actualiza la información ahí mostrada.



Ilustración 56.- Contacto actualizado

Esperar respuesta en Activity

Si queremos que el activity de detalles actualice los datos al recibir la respuesta positiva después de actualizar o guardar los cambios, entonces es necesario una modificación.

Todas las instrucciones encargadas de cargar y mostrar los datos en la interfaz, vamos a moverlas a un método, para que este pueda ser llamado las veces que sea necesario.

La clase DetallesContactoActivity queda como:

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS
;
import mx.edu.itslibres.android.contactosapp.models.Contacto;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;
import android.widget.TextView;
```

```
import android.widget.Toast;

public class DetallesContactoActivity extends Activity {

    //inicializar los elementos de la vista
    TextView NombreLbl;
    TextView ApellidosLbl;
    TextView EmailLbl;
    TextView TelefonoLbl;
    ImageButton EditarBtn;

    ImageButton AgregarContactoBtn;

    AccesoContactosWS accesoContactosWS;
    Contacto contacto;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //asignamos el layout contacto_form
        setContentView(R.layout.detalles_contacto);

        this.AgregarContactoBtn = (ImageButton)
findViewById(R.id.AgregarCImgb);
        this.NombreLbl = (TextView) findViewById(R.id.NombreLbl);
        this.ApellidosLbl = (TextView)
findViewById(R.id.ApellidosLbl);
        this.EmailLbl = (TextView) findViewById(R.id.EmailLbl);
        this.TelefonoLbl =
        (TextView) findViewById(R.id.TelefonoLbl);

        this.EditarBtn =
        (ImageButton) findViewById(R.id.EditarBtn);

        //Recuperar los datos recibidos del activity que invoco
        //si se tienen datos entonces procesarlos
        if(getIntent().getExtras() != null)
        {
            accesoContactosWS = new AccesoContactosWS();

            Bundle extras = getIntent().getExtras();
            int idcontacto = extras.getInt("Id");

            //llamamos al metodo que carga los datos de un contacto
            CargarDatosContacto(idcontacto);
        }
    }
}
```

```
}

//implementamos el evento clic para AgregarContactoBtn
this.AgregarContactoBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // hacer que abra el nuevo layout
        Intent agregarcontacto = new
Intent(DetallesContactoActivity.this,
AregarContactoActivity.class);
        startActivity(agregarcontacto);
    }
});

this.EditarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {

        //Creamos el objeto que ayudara a llevar los
datos
        Bundle datosextras = new Bundle();
        datosextras.putInt("Id",
Integer.valueOf(contacto.getId()));

        Intent editarcontacto = new
Intent(DetallesContactoActivity.this,
EditarContactoActivity.class);
        editarcontacto.putExtras(datosextras);
        //startActivity(editarcontacto);

        //cambiamos la llamada a espera de respuesta
startActivityForResult(editarcontacto, 0);

    }
});

private void CargarDatosContacto(int idContacto)
{
    contacto = accesoContactosWS.DatosContacto(idContacto);
    if(contacto!=null)
    {
```

```
//Cuando se tenga conexion al webservice, se recuperará el objeto y se //mostrarán en la vista

//por ahora mostramos solo el id
this.NombreLbl.setText(contacto.getNombre());
this.ApellidosLbl.setText(contacto.getApellidos());
this.EmailLbl.setText(contacto.getEmail());
this.TelefonoLbl.setText(contacto.getTelefono());

}else
Toast.makeText(getApplicationContext(), "No se pudo obtener los datos del contacto", Toast.LENGTH_SHORT).show();
}

//metodo para controlar la respuesta del intent Editar
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // TODO Auto-generated method stub
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == 0){
        if (resultCode == RESULT_OK) {

            //se recibio respuesta afirmativa, los datos del contacto fueron actualizados
            //entonces procedemos a recargar los datos

            contacto
                //llamamos al metodo que carga los datos del contacto
                CargarDatosContacto(contacto.getId());
        }
        //si no se actualizaron los datos no se hace nada
    }
}
}
```

Al probar la aplicación y tener estos datos del contacto



Ilustración 57.- Datos actuales

Después de presionar Guardar



Ilustración 58.- Datos actualizados

El código del botón Guardar de EditarContactoActivity

```
//al presionar el boton Guardar
this.GuardarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {

        //antes de mandar a guardar, recuperar los
nuevos datos de los Txt

        contacto.setNombre(NombreTxt.getText().toString());

        contacto.setApellidos(ApellidosTxt.getText().toString());

        contacto.setEmail(EmailTxt.getText().toString());

        contacto.setTelefono(TelefonoTxt.getText().toString());

        try{

            if(accesoContactosWS.ActualizarContacto(contacto))
            {
                Toast.makeText(getApplicationContext(),
                "Los datos del contacto han sido guardados",
                Toast.LENGTH_SHORT).show();

                //armamos la respuesta de nuestro
intento
                //dando una respuesta positiva
                Intent intent = new Intent();
                setResult(RESULT_OK, intent);
                finish();
            }
            else
                Toast.makeText(getApplicationContext(),
                "Ocurrió un error al actualizar los datos del contacto",
                Toast.LENGTH_SHORT).show();

                finish();
            }catch(Exception e){
                finish();
            }
        }
    }
})
```

```
});
```

Registrar contacto

Es su turno implementar este activity, es muy similar a Editar.

Recuerde que el método RegistrarContacto de la clase que accede al WS devuelve un objeto de la clase Contacto, por lo tanto, si se recibe el objeto deberá pasar a mostrar los detalles, en caso de recibir null no hacer nada e informar del error al usuario.

Llenando los datos del contacto.



Ilustración 59.- Registro de nuevo contacto

Contacto ya registrado



Ilustración 60.- Contacto registrado

La Clase **AgregarContactoActivity**:

```
package mx.edu.itslibres.android.contactosapp.activities;

import mx.edu.itslibres.android.contactosapp.R;
import
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS
;
import mx.edu.itslibres.android.contactosapp.models.Contacto;
import
mx.edu.itslibres.android.contactosapp.threads.RegistrarContactoThr
ead;
import android.app.Activity;
import android.app.Application;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.Toast;

//agregamos el constructor
public class AgregarContactoActivity extends Activity {
```

```
ImageButton CancelarBtn;
ImageButton GuardarBtn;

EditText NombreTxt;
EditText ApellidosTxt;
EditText EmailTxt;
EditText TelefonoTxt;

AccesoContactosWS accesoContactosWS;
Contacto contacto;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //asignamos el layout contacto_form
    setContentView(R.layout.contacto_form);

    this.CancelarBtn= (ImageButton)
findViewById(R.id.CancelarBtn);
    this.GuardarBtn = (ImageButton)
findViewById(R.id.GuardarBtn);

    this.NombreTxt = (EditText)findViewById(R.id.NombreTxt);

    this.ApellidosTxt=(EditText)findViewById(R.id.ApellidosTxt);
    this.EmailTxt = (EditText)findViewById(R.id.EmailTxt);
    this.TelefonoTxt =
(EditText)findViewById(R.id.TeléfonoTxt);

    this.CancelarBtn= (ImageButton)
findViewById(R.id.CancelarBtn);
    this.GuardarBtn = (ImageButton)
findViewById(R.id.GuardarBtn);

//al presionar el boton cancelar
this.CancelarBtn.setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        //finalizar el intent
        finish();

    }
})
```

```
});  
  
//al presionar el boton Guardar  
this.GuardarBtn.setOnClickListener(new  
View.OnClickListener() {  
  
    @Override  
    public void onClick(View arg0) {  
  
        //inicializar el objeto contacto  
        contacto = new Contacto();  
  
        //antes de mandar a guardar, recuperar los  
nuevos datos de los Txt  
  
        contacto.setNombre(NombreTxt.getText().toString());  
  
        contacto.setApellidos(ApellidosTxt.getText().toString());  
  
        contacto.setEmail>EmailTxt.getText().toString());  
  
        contacto.setTelefono(TelefonoTxt.getText().toString());  
  
        try{  
            //inicializamos el objeto que conecta al  
servicio  
  
AccesoContactosWS();  
  
contacto  
  
el mismo objeto enviado  
contacto  
  
        contacto =  
accesoContactosWS.RegistrarContacto(contacto);  
  
        //si lo recibido es un objeto lleno,  
mandar a mostrar el perfil  
ocurrido  
        //sino informar al usuario del error  
        if(contacto != null)  
        {
```

```
        Toast.makeText(getApplicationContext(),
"El contacto ha sido registrado correctamente",
Toast.LENGTH_SHORT).show();                                //preparamos la llamada al intent
de detalles

        Bundle datosextras = new Bundle();
datosextras.putInt("Id",
contacto.getId());                                         //pasamos a mostrar el activity de
los detalles

        Intent detalles = new
Intent(AgregarContactoActivity.this,
DetallesContactoActivity.class);                           //antes de llamar al intent,
agregamos los extras
detalles.putExtras(datosextras);

        //terminamos este intent y
lanzamos el de detalles
finish();

        startActivity(detalles);

    }
else
        Toast.makeText(getApplicationContext(),
"Ocurrió un error al intentar registrar el nuevo contacto.",
Toast.LENGTH_SHORT).show();

    }catch(Exception e){
        Toast.makeText(getApplicationContext(),
"Ocurrió un error al intentar registrar el nuevo contacto.",
Toast.LENGTH_SHORT).show();
        finish();
    }
}
});                                         }

}
```

El evento onResume()

Ahora si regresamos a la pantalla principal el activity se queda con los datos que recibió primeramente. Queremos entonces que al volver al activity, este actualice los datos mostrados.

Debe implementarse el evento onResume() de la clase Activity en MainActivity (mover el código que carga los contactos de onStart a onResume):

```
@Override  
protected void onResume() {  
    // TODO Auto-generated method stub  
    super.onResume();  
  
    //Toast.makeText(getApplicationContext(), "Regresaste a  
mi",Toast.LENGTH_SHORT).show();  
    try{  
        accesoContactosWS = new AccesoContactosWS();  
        this.contactoslstw.setAdapter(new ContactoAdapter(this,  
accesoContactosWS.ListaContactos()));  
    }catch(Exception e)  
    {  
        Toast.makeText(this, "Ocurrió un error al cargar  
la lista de contactos.", Toast.LENGTH_SHORT).show();  
        //finish();  
    }  
}
```

El Uso de Intents disponibles en Android

El marcado telefónico

Implementar el marcado telefónico para el botón llamar (en detalles de contacto)

Consultar <https://snipt.net/Martin/android-intent-usage/> para ver opciones del uso de Intents

```
this.LlamarBtn.setOnClickListener(new  
View.OnClickListener() {  
  
    @Override  
    public void onClick(View arg0) {  
        // TODO Auto-generated method stub  
  
        //aqui lanzar el intent para llamada  
telefonica
```

```
Intent LanzarLlamada = new  
Intent(Intent.ACTION_CALL);  
  
LanzarLlamada.setData(Uri.parse("tel:"+contacto.getTelefono())  
));  
startActivity(LanzarLlamada);  
  
}  
});
```

Agregamos el permiso para realizar llamadas telefónicas.

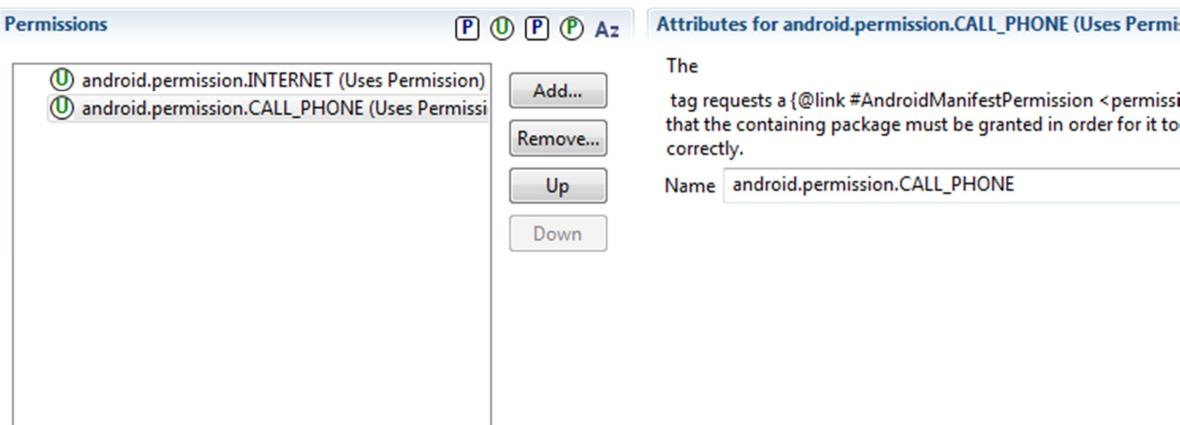


Ilustración 61.- Agregar permiso de llamada

Al correr la aplicación y presionar Llamar, se abrirá el marcador de teléfono.

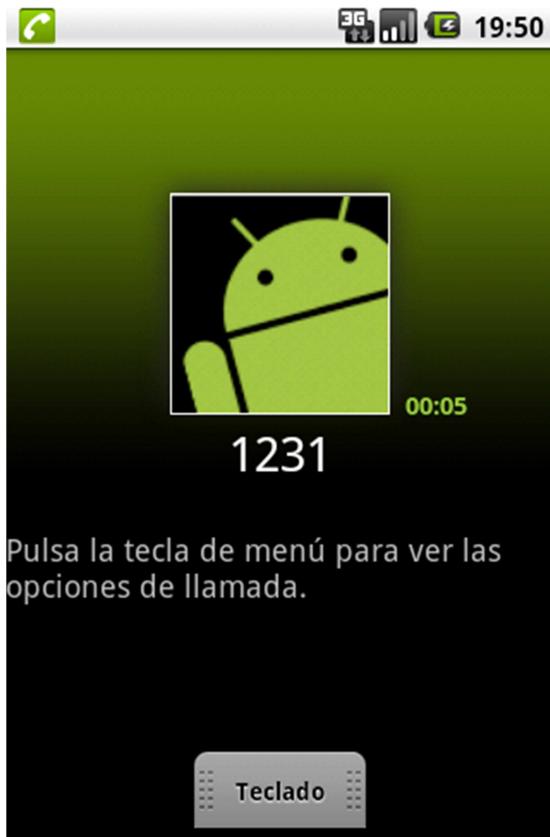


Ilustración 62.- Marcador telefónico en función

El botón email

El botón email en DetallesContactoActivity quedara programado como:

```
this.EnviaEmailBtn.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        Uri uri =
Uri.parse("mailto:"+contacto.getEmail());
        Intent emailIntent =
Intent(Intent.ACTION_SENDTO, uri);

        startActivityForResult(Intent.createChooser(emailIntent, "Email para
"+contacto.getEmail()));
    }
});
```

Corriendo la aplicación en Android 4.0.3

En versiones superiores a Android no funciona el acceso a la red directamente desde el Hilo (Subproceso) principal, para ello se debe utilizar la clase AsyncTask para crear hilos independientes.

Para arreglar el proyecto, agreguemos un nuevo subpaquete llamado threads.

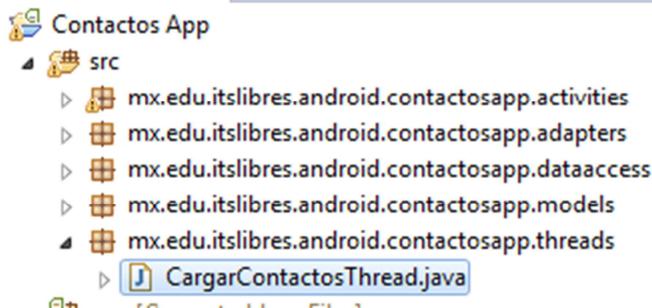


Ilustración 63.- nuevo subpaquete

Implementando Hilos o Threads para el acceso a través de la red

Lista de clases agregadas para delegarles el acceso al Servicio Web. Debe ser una por acción a invocar.

La clase CargarContactosThread

```
package mx.edu.itslibres.android.contactosapp.threads;

import java.util.List;

import
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS
;
import mx.edu.itslibres.android.contactosapp.models.Contacto;

import android.os.AsyncTask;

public final class CargarContactosThread extends AsyncTask<Void,
Void, List<Contacto>> {

    AccesoContactosWS           accesoContactosWS      =      new
AccesoContactosWS();

    @Override
    protected final List<Contacto> doInBackground(Void... arg0) {
        // TODO Auto-generated method stub
        //return null;
        return accesoContactosWS.ListaContactos();
```

```
    }  
}
```

La clase **CargarContactoThread**

```
package mx.edu.itslibres.android.contactosapp.threads;  
  
import  
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS  
;  
import mx.edu.itslibres.android.contactosapp.models.Contacto;  
import android.os.AsyncTask;  
  
public final class CargarContactoThread extends AsyncTask<Integer,  
Void, Contacto> {  
  
    AccesoContactosWS      accesoContactosWS      =      new  
AccesoContactosWS();  
  
    @Override  
    protected final Contacto doInBackground(Integer... arg0) {  
        // TODO Auto-generated method stub  
        return accesoContactosWS.DatosContacto(arg0[0]);  
    }  
}
```

La clase **GuardarContactoThread**

```
package mx.edu.itslibres.android.contactosapp.threads;  
  
import  
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS  
;  
import mx.edu.itslibres.android.contactosapp.models.Contacto;  
import android.os.AsyncTask;  
  
public final class GuardarContactoThread extends  
AsyncTask<Contacto, Void, Boolean> {  
  
    AccesoContactosWS      accesoContactosWS      =      new  
AccesoContactosWS();  
  
    @Override  
    protected final Boolean doInBackground(Contacto... arg0) {  
        // TODO Auto-generated method stub  
        return accesoContactosWS.ActualizarContacto(arg0[0]);  
    }  
}
```

```
    }  
}
```

La clase **RegistrarContactoThread**

```
package mx.edu.itslibres.android.contactosapp.threads;  
  
import  
mx.edu.itslibres.android.contactosapp.dataaccess.AccesoContactosWS  
;  
import mx.edu.itslibres.android.contactosapp.models.Contacto;  
import android.os.AsyncTask;  
  
public class RegistrarContactoThread extends AsyncTask<Contacto,  
Void, Contacto> {  
  
    AccesoContactosWS           accesoContactosWS      =      new  
AccesoContactosWS();  
  
    @Override  
    protected Contacto doInBackground(Contacto... arg0) {  
        // TODO Auto-generated method stub  
        return accesoContactosWS.RegistrarContacto(arg0[0]);  
    }  
}
```

En los Activities donde se invocaba el llamado al WS, ahora se invocara a estas nuevas clases de la forma:

Adaptando las llamadas al WS a los nuevos Threads

En **MainActivity** donde se tenía:

```
this.contactoslstw.setAdapter(new  
accesoContactosWS.ListaContactos()));  
ContactoAdapter(this,
```

Cambiar a:

```
this.contactoslstw.setAdapter(new  
ContactoAdapter(this,      new  
CargarContactosThread().execute().get()));  
ContactoAdapter(this,
```

En **DetallesContactoActivity** donde se tenía:

```
contacto = accesoContactosWS.DatosContacto(idContacto);
```

Cambiar a:

```
contacto = new CargarContactoThread().execute(idContacto).get();
```

En **EditarContactoActivity** donde se tenía:

```
contacto = accesoContactosWS.DatosContacto(idcontacto);  
y
```

```
if(accesoContactosWS.ActualizarContacto(contacto))
```

Cambiar a:

```
contacto = new CargarContactoThread().execute(idcontacto).get();  
y
```

```
if(new GuardarContactoThread().execute(contacto).get())
```

En **AgregarContactoActivity** donde se tenía:

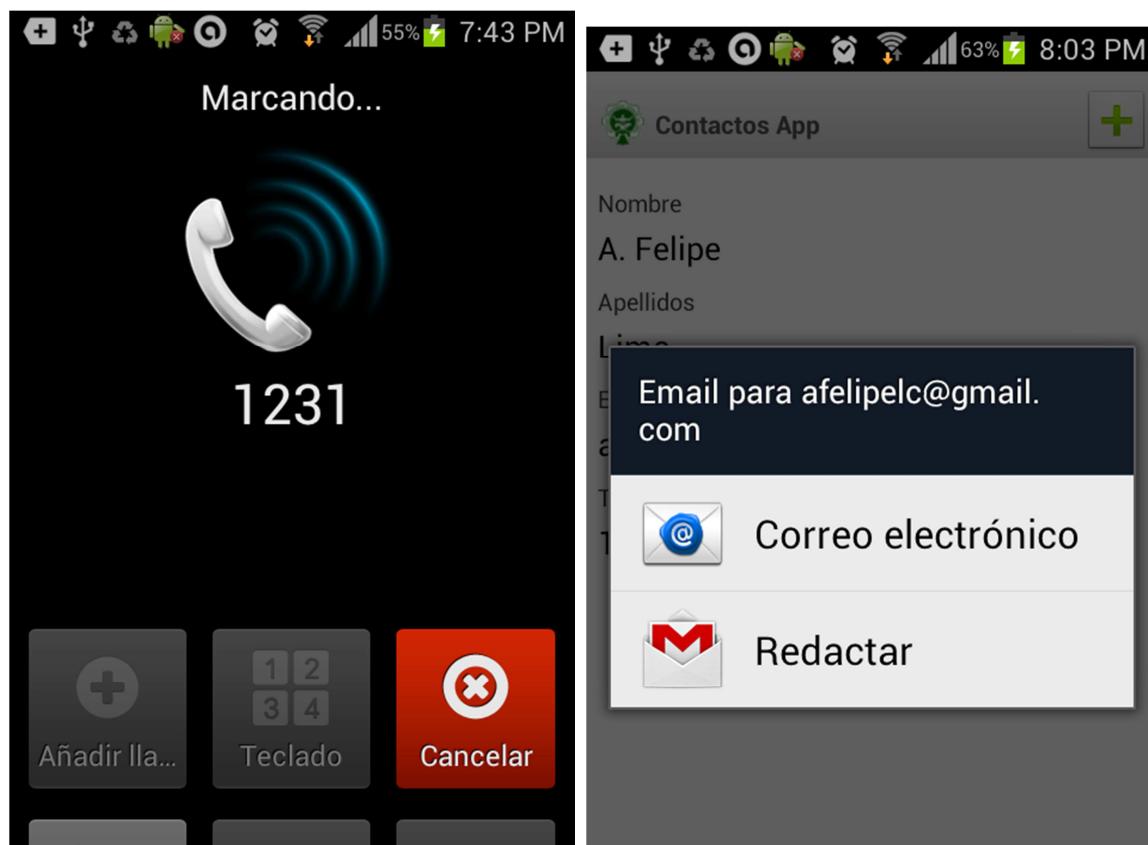
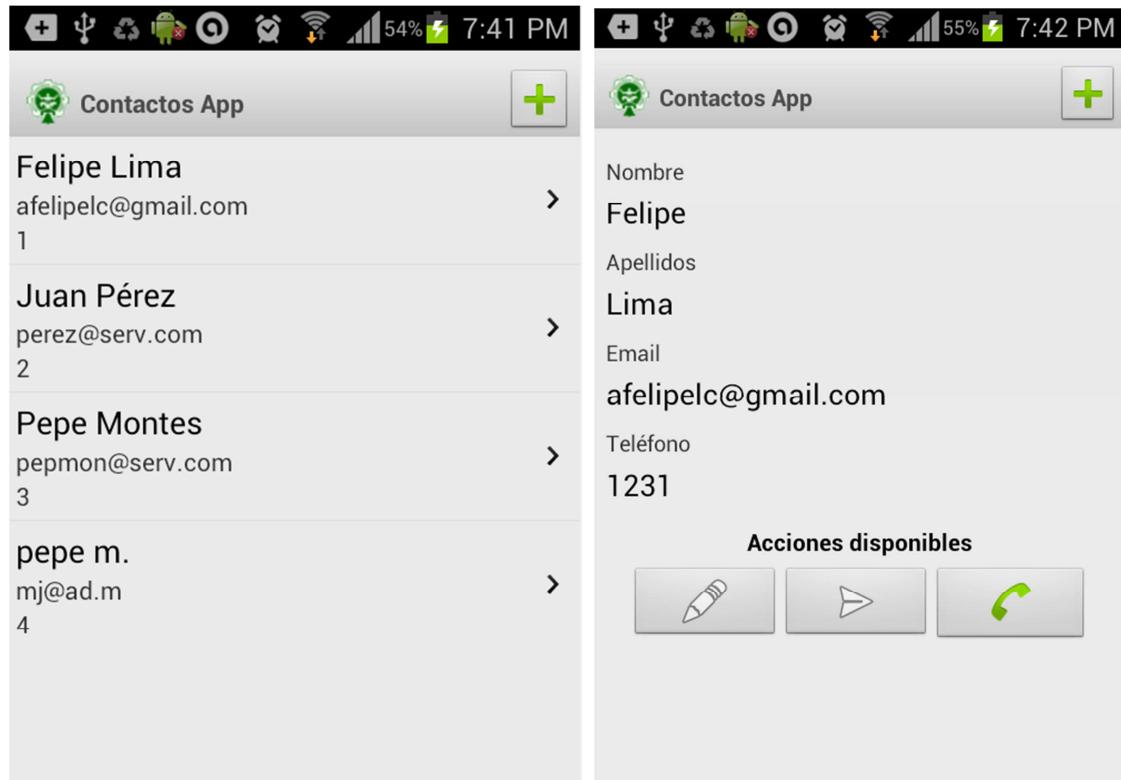
```
contacto = accesoContactosWS.RegistrarContacto(contacto);
```

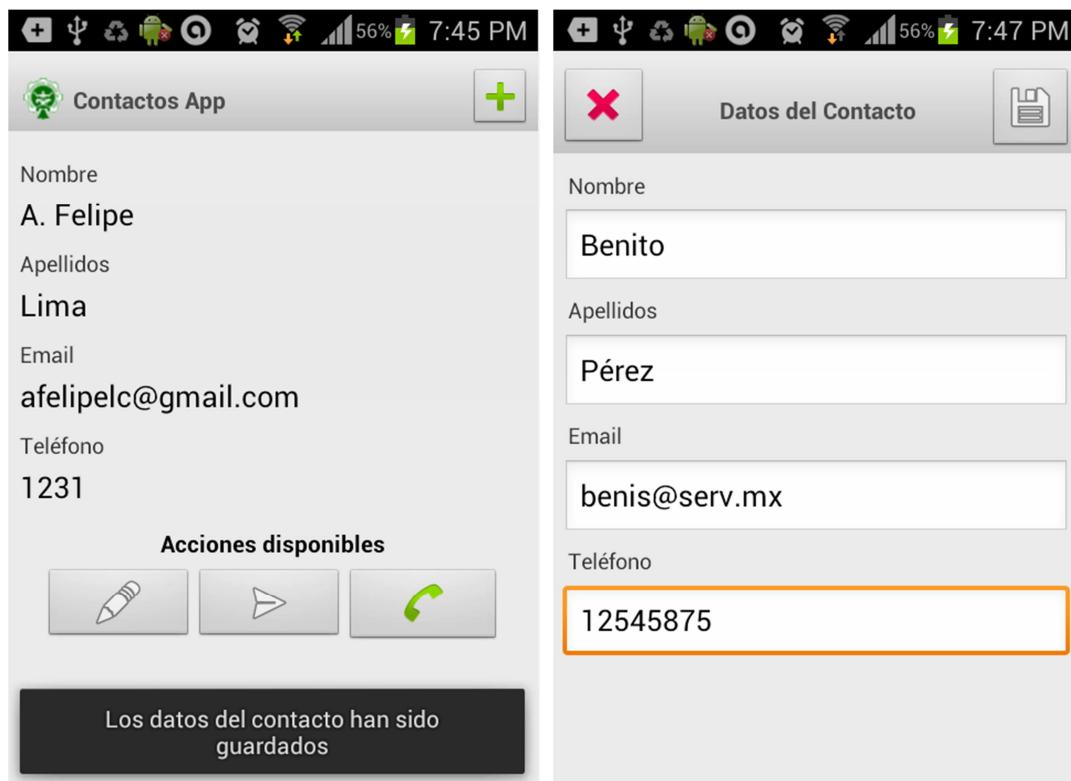
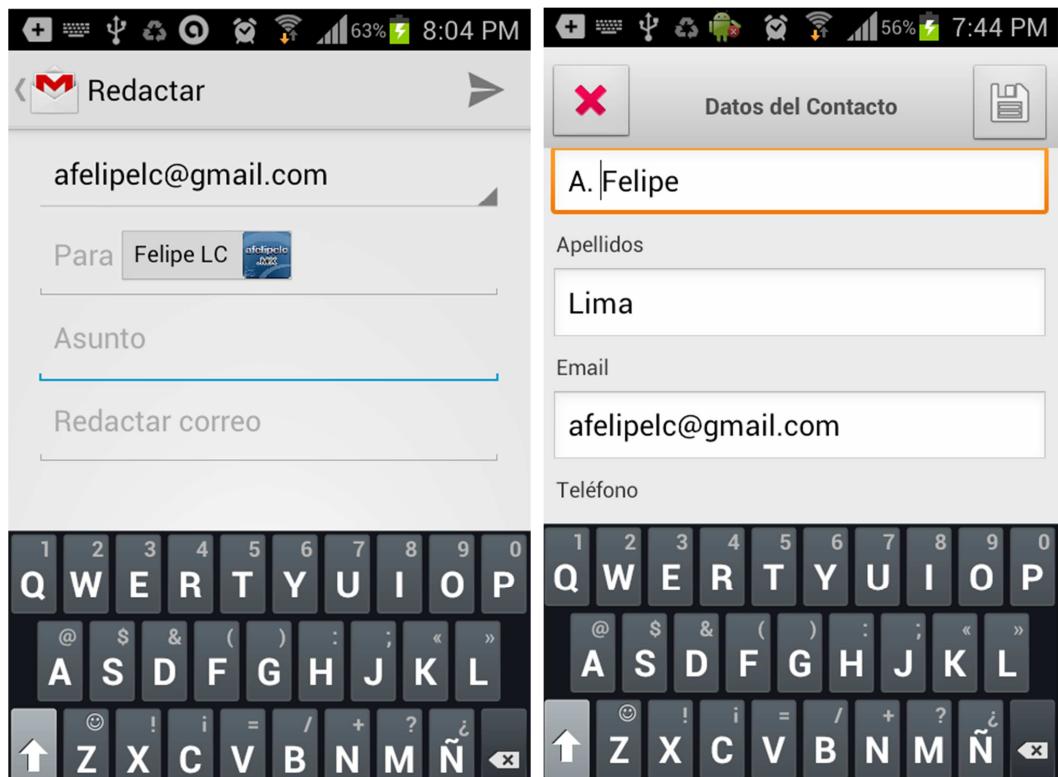
Cambiar a:

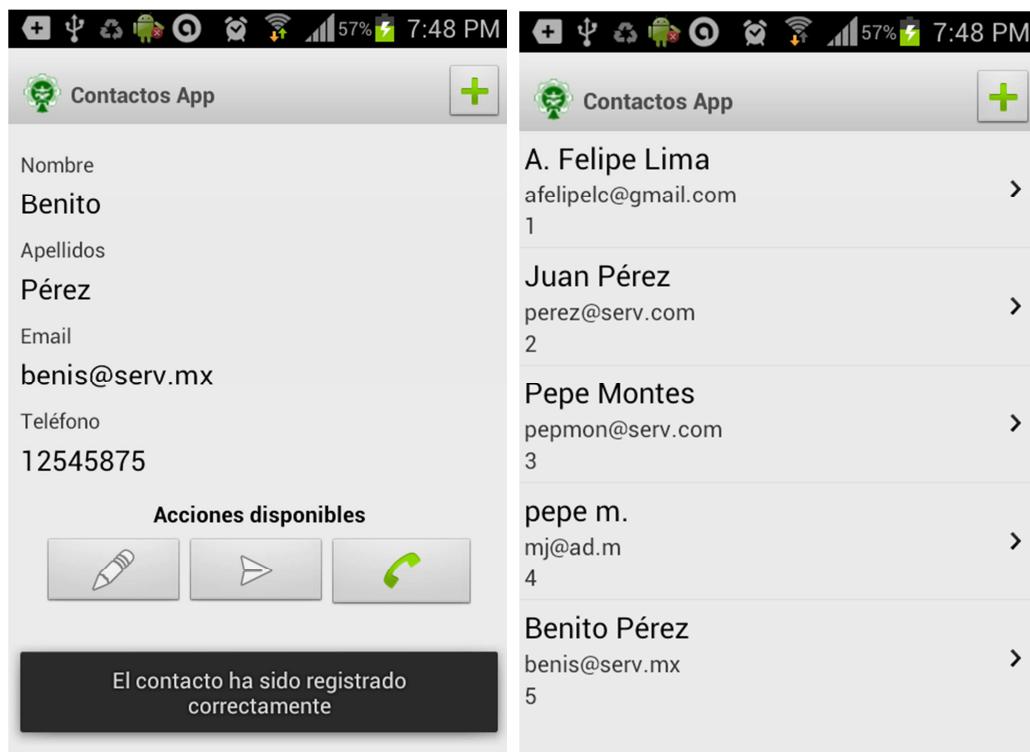
```
contacto = new RegistrarContactoThread().execute(contacto).get();
```

El resultado en Android ICS 4.0.3

Estas son las capturas de pantalla tomadas de la aplicación ya en funcionamiento en un teléfono con la versión de Android ICS 4.0.3







Últimos detalles en la aplicación

Ocultar elementos de la UI

Hace falta ahora ocultar el ID de la lista (editar el layout list_item_contacto).

Seleccionar el TextView, y en propiedades establecer GONE en Visibility:

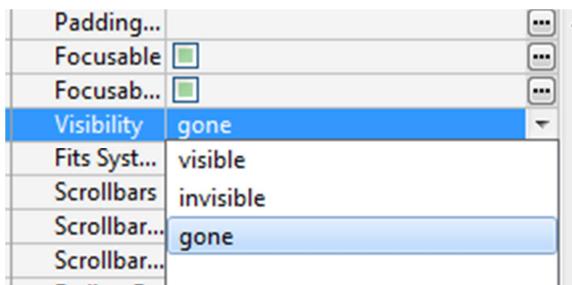


Ilustración 64.- Ocultar elemento de la UI

El resultado final será

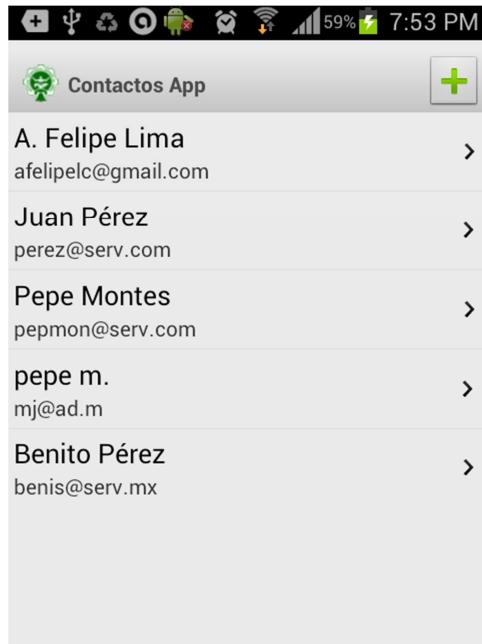


Ilustración 65.- Elemento oculto

Mensajes de confirmación

Agregar un mensaje para preguntar al usuario si se marca al teléfono o no:

Agregamos dentro de onCreate() del activity Detalles

```
//Dialogo Yes NO para preguntar si marca al telefono o no
final AlertDialog.Builder confirmarLlamada = new
AlertDialog.Builder(this);
confirmarLlamada.setMessage("¿Confirma llamar a
"+contacto.getNombre() + " " + contacto.getApellidos() + " ?")
.setPositiveButton("Si",
new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface,
int i) {
        //Si dijo que si, entonces llamar

        //aqui lanzar el intent para llamada telefonica
        Intent LanzarLlamada = new
Intent(Intent.ACTION_CALL);

        LanzarLlamada.setData(Uri.parse("tel:"+contacto.getTelefono()
));
        startActivity(LanzarLlamada);
    }
}
```

```
        })
        .setNegativeButton("Cancelar",
new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface,
int i) {
        //no hacer nada
    }
});
```

El botón llamar quedará como:

```
this.LlamarBtn.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        //aqui lanzar el intent para llamada
        confirmarLlamada.show();
    }
});
```

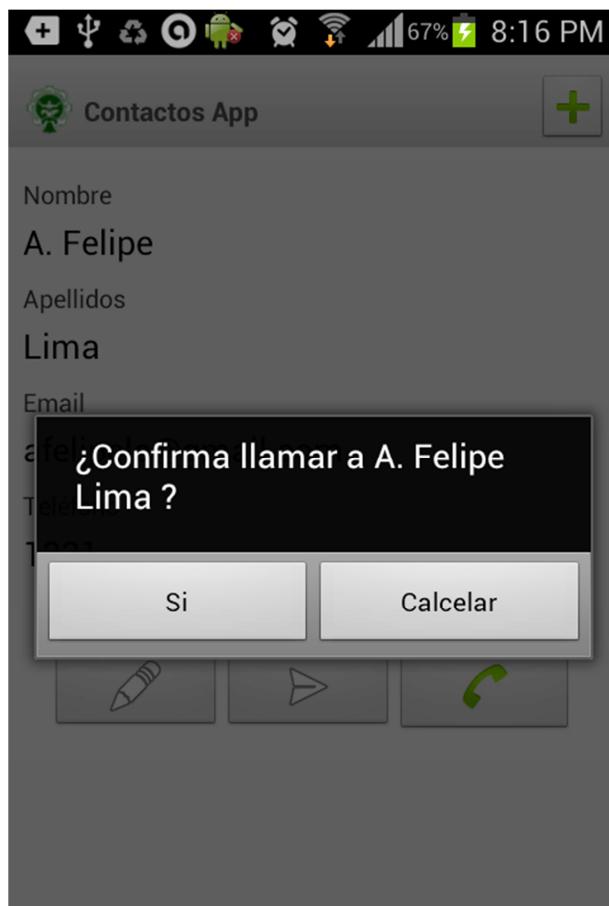


Ilustración 66.- Función del diálogo

Recursos utilizados en el proyecto

Al momento de escribir este documento no había publicado los recursos, sin embargo, accede a www.afelipelc.mx donde puedes buscar un artículo relacionado a este documento, ahí encontrarás los recursos: código fuente.

Conclusión

Desarrollar una aplicación para Android puede parecer complejo cuando se es novato, sin embargo, a pesar de leer mucho y poner en práctica los conocimientos acerca de esta plataforma nos parecerá poco ya que hay bastantes cosas que aprender de esto.

Sin embargo, con este ejemplo destinado a un taller con duración de 10 horas ha sido mucho lo aprendido; en resumen con este ejemplo se ha aprendido a:

- Preparar el entorno de desarrollo
- Crear proyectos Android
- Diseñar interfaces

- Manipular los elementos de las UI
- Acceder a Web Services
- Manipular datos a través de Web Services
- Lanzar Intents de Android
- Implementar Hilos o Threads
- Implementar cuadros de diálogo

Hasta aquí hemos terminado nuestra aplicación.

Gracias por su atención y sobre todo, su esfuerzo!

Atte.

Felipe Lima Cortés

afelipelc@gmail.com

afelipelc.wordpress.com

Noviembre de 2012

