

Desarrollo de aplicaciones e introducción a juegos con Android Studio y AndEngine.

Tema 6: Servicios Web en Android



Jose Manuel Fornés Rumbao

Departamento de Ingeniería Telemática
Universidad de Sevilla

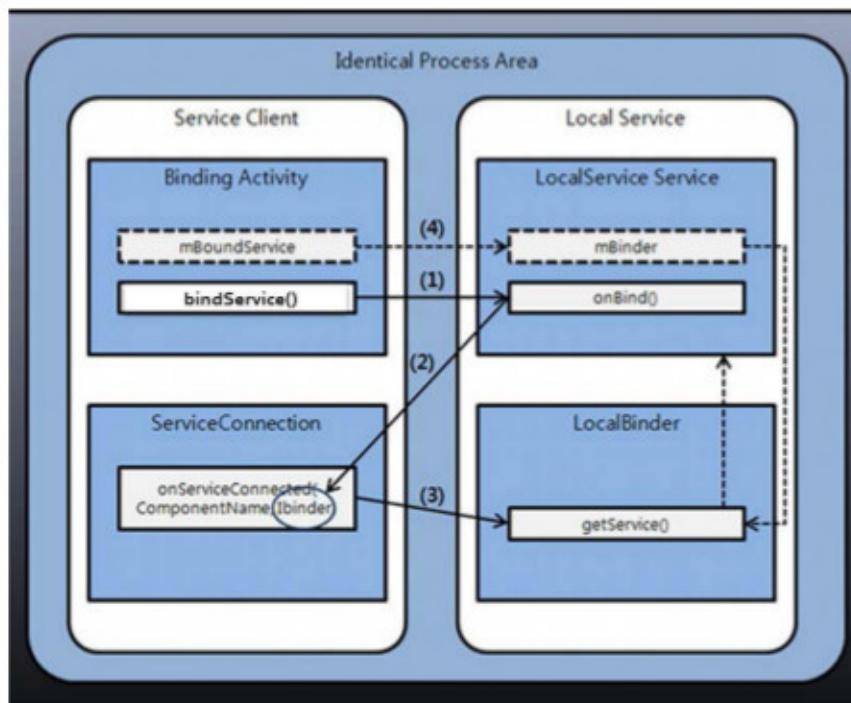
jfornes@us.es

Índice

1. Introducción
2. AsyncTask
3. Sockets
4. Clientes HTTP
 1. HttpClient
 2. HttpURLConnection
5. Volley
6. Servicios Web
 1. SOAP
 2. SOAP en Android
 3. REST
 4. REST en Android

Introducción

- Android ofrece muchos mecanismos para la comunicación entre procesos de forma local
- También muchas aplicaciones Android ofrecen y usan servicios y datos de Internet
- La programación robusta, extensible y eficiente de aplicaciones que usen la red es una tarea compleja ya que debe tratar con situaciones que aplicaciones no conectadas no necesitan



Introducción

- Android incluye muchas clases para la programación en red:
 - java.net (socket, URL)
 - org.apache (HttpRequest, HttpResponse)
 - android.net (URI, AndroidHttpClient, AudioStream)
- Para permitir el acceso a internet a una aplicación hay que añadirle los permisos en el fichero AndroidManifest.xml

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

- Hay 3 clases principales en Java Sockets
 - ServerSocket: Representa al lado servidor del socket que espera conexiones entrantes del cliente
<http://developer.android.com/reference/java/net/ServerSocket.html>
 - Socket: Ofrece el socket TCO del lado del cliente
<http://developer.android.com/reference/java/net/Socket.html>
 - InetAddress: Define la dirección del protocolo IP
<http://developer.android.com/reference/java/net/InetAddress.html>

AsyncTask

- Todas las tareas que puedan introducir retardos no pueden ser ejecutadas en el hilo principal del UI ya que Android no puede perder el control del UI
- AsyncTask es un mecanismo de Android creado para ayudar a manejar operaciones que necesiten mucho tiempo y que deban reportar al hilo del UI
- Para ello debemos crear una clase que herede de AsyncTask y que implemente los métodos:
 - onPreExecute(): Se llama al crearse el objeto AsyncTask y se ejecuta sobre el hilo principal. Cuando termina, crea un hilo secundario y ejecuta su trabajo dentro de doInBackground():
 - onProgressUpdate(): Durante la ejecución del hilo en segundo plano se pueden hacer llamadas al hilo principal con este método con ayuda de publishProgress()
 - onPostExecute(): Se ejecuta al terminar de ejecutarse doInBackground() y se termina el hilo secundario
 - cancel(): Para cancelar la tarea en segundo plano



Sockets

- El uso de los sockets es mediante flujos de entrada y salida en Java

```
InputStream in = someSocket.getInputStream();
Reader reader = new InputStreamReader(in);
for (int data; (data = reader.read()) != -1; ) {
    char theChar = (char) data;
    // ... do something with the data
}
reader.close();
```

Socket->flujo bytes->flujo
caracteres: Leemos caracteres

```
BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(someSocket.getInputStream()));
for (String data; (data = bufferedReader.readLine()) != null; ){
// ... do something with the data
}
bufferedReader.close();
```

Socket->flujo bytes->flujo
caracteres->buffer: Leemos del
buffer

```
OutputStreamWriter out = new OutputStreamWriter
(someSocket.getOutputStream());
String string1 = "Socket IO", string2 = " is fun";
out.write(string);
out.append(string);
out.flush();
out.close();
```

Escribimos caracteres->flujo
bytes->Socket

Sockets

- Los sockets, debido a los retardos de la red, pueden provocar la pérdida del control del UI por parte de Android por lo que deben hacerse en segundo plano

```
public class MainActivity extends Activity {  
    TextView mTextView;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        mTextView= (TextView) findViewById(R.id.mitexto);  
        new MiHttpGet().execute("www.google.es");  
    }  
    private void onFinishGetRequest(String result) {  
        mTextView.setText(result);  
    }  
    private class MiHttpGet extends AsyncTask<String, Void, String> {  
        protected String doInBackground(String... params) {  
            Socket socket = null;  
            StringBuffer data = new StringBuffer();  
            try {  
                socket = new Socket(params[0], 80);  
                PrintWriter pw = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()), true);  
                pw.println("GET /index.html");  
                BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
                String rawData;  
                while ((rawData = br.readLine()) != null) {  
                    data.append(rawData);  
                }  
            } catch( IOException e) {}  
            return data.toString(); // close socket ...  
        }  
        protected void onPostExecute(String result) {  
            onFinishGetRequest(result);  
        }  
    }  
}
```

Clientes HTTP

- Android incluye dos clientes HTTP: HttpURLConnection y Apache HTTP Client
- Ambos soportan HTTPS, streaming, descargas y subidas, temporizadores programables, IPv6, etc
- Para versiones de Android 2.2 (Froyo) y anteriores es mejor usar HTTP Client.
- En cambio para versiones superiores la mejor opción es HttpURLConnection:
 - Es una API simple y de pequeño tamaño, ajustado para Android
 - El uso de la red es reducido porque soporta cache y compression transparente, mejorando la velocidad y el ahorro de batería

HttpURLConnection

- Se usa para enviar y recibir datos HTTP de cualquier tipo y longitud sobre la web
- Esta clase también se puede utilizar para enviar y recibir datos en streaming cuya longitud no se conoce por adelantado
- Hace uso de la clase URL que permite parsear un URL y extraer sus componentes (protocolo, host, puerto, URI, etc)
- Tras obtener un HttpURLConnection llamando a `URL.openConnection()` podemos
 1. Leer el código de estado: 200, 301, 304, etc.
 2. Establecer las cabeceras de la petición: Accept, Referer, User-Agent, etc.
 3. Leer las cabeceras de respuesta: Content-Type, Location, etc.
 4. Uso de un CookieManager en lugar de cabeceras Raw
- Luego, usando el método: `openInputStream()` obtenemos el flujo de lectura. Puede encapsularse en un `InputStreamReader`, o `BufferedReader` para usar `readLine`
- Cuando se ha leído el cuerpo de la respuesta, se debería cerrar el HttpURLConnection mediante la llamada a `disconnect()`, liberando los recursos mantenidos por la conexión.

HttpURLConnection

- HttpURLConnection, debido a los retardos de la red, pueden provocar la pérdida del control del UI por parte de Android por lo que deben hacerse en segundo plano

```
public class MainActivity extends Activity {
    TextView mTextView = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView= (TextView) findViewById(R.id.mitexto);
        new TareaHttpGet().execute("http://www.google.es");
    }
    private void onFinishGetRequest(String result) {
        mTextView.setText(result);
    }
    private class TareaHttpGet extends AsyncTask<String, Void, String> {
        protected String doInBackground(String... params) {
            StringBuffer data = new StringBuffer();
            try {
                String rawData;
                URL url = new URL(params[0]);
                HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
                urlConnection.setRequestMethod("GET");
                BufferedReader in = new BufferedReader(new InputStreamReader
                        (urlConnection.getInputStream()));
                while ((rawData = in.readLine()) != null) {
                    data.append(rawData);
                }
            } catch (Exception e) {}
            return data.toString();
        }
        protected void onPostExecute(String result) {
            onFinishGetRequest(result);
        }
    }
}
```

Volley

- Es una librería HTTP de Google que no viene integrada en el SDK
- Volley surge de que simplemente con HttpURLConnection y AsyncTask, el proceso de conexión y descarga de información se hace engorroso y el control de errores muy complicado, dejando al desarrollador la gestión de temporizadores, reintentos, etc.
- Las llamadas a la red mediante Volley son asíncronas por lo que no tendremos que gestionar las tareas en segundo plano. Volley lo hace por nosotros.
- Las peticiones se gestionan por una cola con prioridades y no en serie como antes
- Gestiona una cache y la memoria. Cabe recordar que en un cambio de orientación de la pantalla, en una aplicación sin cache hay que recargar todo de nuevo
- La librería se puede extender y personalizar a nuestro gusto
- Las peticiones de la cola se pueden cancelar
- La gestión de errores y depuración es muy avanzada
- Sin embargo, se debe evitar para la descarga de datos muy pesados ya que la salvaguarda en caché haría el proceso muy lento

Volley

- Puesto que es una librería que no viene en el SDK, hay que integrarla en nuestro proyecto de forma explícita. En Android Studio se hace añadiendo la dependencia en Gradle en el archivo build.gradle del módulo y sincronizamos al terminar

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:21.0.3'  
    compile 'com.mcxiaoke.volley:library:1.0.+'  
}
```

- Es recomendable tener sólo una cola de peticiones por aplicación. Para ello, se crea en la clase Application que nos funciona como un patrón Singleton de la aplicación

```
public class VolleyApplication extends Application {  
    private static VolleyApplication sInstance;  
    private RequestQueue mRequestQueue;  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        mRequestQueue = Volley.newRequestQueue(this);  
        sInstance = this;  
    }  
    public synchronized static VolleyApplication getInstance() {  
        return sInstance;  
    }  
    public RequestQueue getRequestQueue() {  
        return mRequestQueue;  
    }  
}
```

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

Volley

- Para usar la cola RequestQueue se crea una petición Request y se añade a la cola mediante add()
- Las peticiones tienen un Listener con el método onResponse() para la respuesta y un Listener para los errores con el método onError()
- Existen varios tipos de peticiones ya establecidas aunque podemos crear las nuestras propias:
 - StringRequest: Este es el tipo más común, ya que permite solicitar un recurso con formato de texto plano, como son los documentos HTML.
 - ImageRequest: Como su nombre lo indica, permite obtener un recurso gráfico alojado en un servidor externo.
 - JsonObjectRequest: Obtiene una respuesta de tipo JSONObject a partir de un recurso con este formato.
 - JSONArrayRequest: Obtiene como respuesta un objeto del tipo JSONArray a partir de un formato JSON.
- Cada petición tiene sus propios parámetros
- Para crear peticiones personalizadas, se extiende la clase Request<T>, la cual es una plantilla que requiere un tipo T referente a la respuesta que tendrá la petición. Por ejemplo, ImageRequest se extiende de la implementación Request<Bitmap>, debido a que después del parsing se obtiene una respuesta Bitmap.

Volley

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.VolleyLog;
import com.android.volley.toolbox.StringRequest;

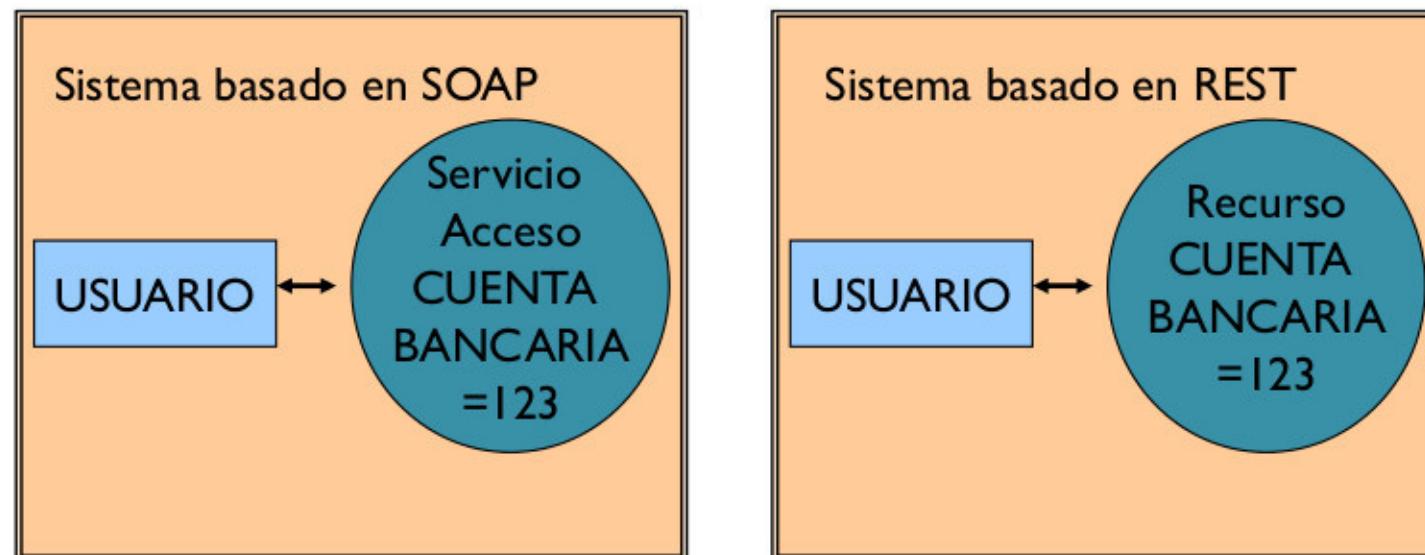
public class MainActivity extends Activity {
    private TextView mTextView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = (TextView) findViewById(R.id.text1);
        final String URL = "http://www.google.es";
        StringRequest request = new StringRequest(URL, new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                mTextView.setText(response.toString());
                VolleyLog.v("Response:%n %s", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                mTextView.setText(error.toString());
                VolleyLog.e("Error: ", error.getMessage());
            }
        });
        // add the request object to the queue to be executed
        VolleyApplication.getInstance().getRequestQueue().add(request);
    }
}
```

Servicios web

- Los Servicios Web son la piedra angular de cualquier desarrollo de sistemas distribuidos actual
- Los ordenadores hablan unos a otros a través de la web usando HTTP y otros protocolos.
- Un servicio web no tiene interfaz gráfica
- Provee una API de métodos que pueden ser invocados en la web, o de recursos accesibles mediante el protocolo HTTP
- Diseñados para proveer “servicios”
- Grandes sitios Web ofrecen este tipo de servicios para que las aplicaciones puedan acceder a los contenidos de estas Webs: Amazon, Google, Yahoo, Foursquare, etc.
- Existen repositorios de servicios, donde muchos son gratuitos:
 - www.webservicesx.net, www.programmableweb.com

Servicios web

- Existen dos enfoques distintos para los servicios web, SOAP orientado a métodos y REST orientado a recursos



SOAP

- Los servicios web SOAP han dado lugar a un nuevo modo de diseñar sistemas distribuidos:
 - Arquitecturas SOA (Service Oriented Architecture)
- SOA = colección de servicios
 - Más información en <http://www.service-architecture.com/>
- Son un estándar basado en protocolos abiertos como HTTP y SOAP
 - SOAP es un vocabulario XML que representa RPCs
 - <http://www.w3.org/TR/SOAP>
- Servicio web = aplicación que:
 - se ejecuta en un servidor web
 - expone métodos a clientes
 - escucha peticiones HTTP representando comandos que invocan a métodos Web
 - ejecuta métodos web y devuelve resultados

SOAP

- SOAP es un protocolo de comunicación basado en XML útil para la comunicación entre aplicaciones
 - <http://www.w3.org/2000/xp/Group/>
- SOAP es un mecanismo para el intercambio de mensajes a través de Internet independiente de los lenguajes de programación
 - Es un protocolo de transporte
- Los clientes envían una petición SOAP mediante un HTTP POST normalmente y reciben un código de respuesta (éxito o error) y una respuesta SOAP
- Una petición SOAP tiene fundamentalmente 3 partes:
 - Envoltorio: Define el marco para definir lo que va dentro del mensaje y cómo procesarlo
 - Reglas: Incluye un conjunto de reglas codificadas para expresar instancias de los tipos de datos definidos en la aplicación
 - Convención: Usado para representar las llamadas a procedimientos remotos y sus respuestas

SOAP

- Un mensaje SOAP es un mensaje XML que consta de un conjunto de cabeceras opcionales y de un cuerpo.



```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
      <getProductDetailsResult>
        <productName>Toptimate 3-Piece Set</productName>
        <productId>827635</productId>
        <description>3-Piece luggage set. Black Polyester.</description>
        <price>96.50</price>
        <inStock>true</inStock>
      </getProductDetailsResult>
    </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP. Android

- En Android existen muchas librerías para el desarrollo de clientes para servicios web basados en SOAP
- Con independencia de los mecanismos de conexión, necesitamos un parser XML para que interprete el XML de respuesta y obtengamos los resultados.
- La librería más comúnmente usada en Android es Ksoap2-android. Es una librería cliente SOAP de poco peso y eficiente.
- Para usarla:
 1. Incorporamos la librería ksoap2-android-assembly-3.4.0-jar-with-dependencies.jar, en la carpeta libs del proyecto que debemos crear



2. Modificamos las dependencias de Gradle en el archivo build.gradle del módulo para que se compile con el proyecto

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:21.0.3'  
    compile files('libs/ksoap2-android-assembly-3.4.0-jar-with-dependencies.jar')  
}
```

SOAP.Android

3. Se definen cuatro variables tipo String con los parámetros obtenidos del WDSL del servicio y de la URL

```
String NAMESPACE = "..."; StringURL = "..."; String SOAP_ACTION = "..."; String METODO = "...";
```

4. Crear un objeto de tipo SoapObject e inicializarlo con las variables NAMESPACE y METODO como argumentos:

```
SoapObject Solicitud = new SoapObject(NAMESPACE, METODO);
```

5. Si el servicio web necesita parámetros (no siempre), se añaden los parámetros al SoapObject creado anteriormente, indicando su nombre y su valor

```
Solicitud.addProperty ("nombre parámetro1","valor parámetro1");  
Solicitud.addProperty ("nombre parámetro2","valor parámetro2");
```

6. Creamos el envoltorio

```
SoapSerializationEnvelope Envoltorio =new SoapSerializationEnvelope (SoapEnvelope.VER11);
```

7. Asignamos el SoapObject al envoltorio, especificando si es un servicio .Net

```
Envoltorio.dotNet = true; /*6*/ Envoltorio.dotNet = false;  
Envoltorio.setOutputSoapObject (Solicitud);
```

8. Se crea un objeto tipo HttpTransportSE para que haga la conexión y se inicializa con la URL como parámetro:

```
HttpTransportSE TransporteHttp = new HttpTransportSE(URL);
```

9. Se envía la petición al servicio mediante el método call del transporte

```
TransporteHttp.call (SOAP_ACTION, Envoltorio);
```

10. Se procesa la respuesta mediante el método getResponse del envoltorio, según el tipo devuelto

```
String CadenaDevuelta =(String) envelope.getResponse(); /*Por ejemplo String*/
```

SOAP.Android

```
import java.io.IOException;
import org.ksoap2.SoapEnvelope;
import org.ksoap2.SoapFault;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpResponseException;
import org.ksoap2.transport.HttpTransportSE;
import org.xmlpull.v1.XmlPullParserException;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import android.app.Activity;
public class MainActivity extends Activity {
    private static final String SOAP_ACTION = "http://footballpool.dataaccess.eu/TopGoalScorers";
    private static final String METHOD_NAME = "TopGoalScorers";
    private static final String NAMESPACE = "http://footballpool.dataaccess.eu";
    private static final String URL = "http://footballpool.dataaccess.eu/data/info.wso?WSDL";
    private TextView tv;
    private String response;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv= (TextView)findViewById(R.id.txt2);
        myAsyncTask myRequest = new myAsyncTask();
        myRequest.execute();
    }
    private class myAsyncTask extends AsyncTask<Void, Void, Void> {
        @Override
        protected void onPostExecute(Void result) {
            super.onPostExecute(result);
            tv.setText(response);
        }
        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }
        @Override
```

SOAP.Android

```
protected Void doInBackground(Void... arg0) {
    SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
    request.addProperty("iTopN", "5");
    SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
    envelope.setOutputSoapObject(request);
    HttpTransportSE httpTransport = new HttpTransportSE(URL);
    httpTransport.debug = true;
    try {
        httpTransport.call(SOAP_ACTION, envelope);
    } catch (HttpServletResponseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (XmlPullParserException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } //send request
    SoapObject result = null;
    try {
        result = (SoapObject)envelope.getResponse();
    } catch (SoapFault e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    Log.d("App",""+result.getProperty(1).toString());
    response = result.getProperty(1).toString();
    return null;
}
}
```

REST

- REST es una representación de cambio de estado de los recursos
- Es un estilo de arquitectura de software
- Está basado en que la red de páginas web es una máquina de estados virtual donde el usuario avanza a través de una aplicación seleccionando enlaces (transiciones de estado), resultando una página siguiente (que representa el siguiente estado de la aplicación) que se transfiere al usuario y se presenta para su uso
- No es un estándar aunque usa estándares: HTTP, URL, XML/HTML/JSON/JPEG

REST

- REST tiene una serie de restricciones:
 - Cliente-Servidor:
 - Principio de separación con componentes independientes
 - Sin estado:
 - El cliente es el que mantiene una sesión si la necesita
 - Garantiza la visibilidad, fiabilidad y escalabilidad
 - Cacheable:
 - Las respuestas pueden cachearse
 - Es más eficiente pero reduce la fiabilidad
 - Sistema de capas
 - Facilita la escalabilidad del sistema
 - Interfaz uniforme
 - Muy simple

REST

- REST está orientado a recursos y utiliza un identificador para acceder a los recursos: URI
- Para acceder a los recursos representados por su URI, simplemente utiliza los comandos HTTP: GET, POST, PUT y DELETE

HTTP	Método	CRUD	Desc.
POST	CREATE	Crear	-
GET	RETRIEVE	Obtener	No cambia el estado del servidor, misma respuesta siempre
PUT	UPDATE	Actualizar	misma respuesta siempre
DELETE	DELETE	Borrar	misma respuesta siempre

REST.Android

- En Android existen muchas librerías para el desarrollo de clientes para servicios web basados en REST tanto incluidos en el SDK de como externos.
- Con independencia de los mecanismos de conexión, necesitamos un parser XML o JSON para que interprete el XML/JSON de respuesta y obtengamos los resultados.

Petición :

```
http://www.mailingbird.com/api/json/john.doe79@example.com
```

Respuesta :

```
{  
    "ResponseCode": 200,  
    "ResponseStatus": "Request ok",  
    "Result": {  
        "Address": {  
            "SubmittedSyntax": "John.Doe79@hozmail.com",  
            "SanitizedSyntax": "john.doe79@hozmail.com",  
            "CorrectedSyntax": "john.doe79@hotmail.com",  
            "KnownMailer": true  
        },  
        "Lookups": {  
            "Trashmail": false  
        }  
    }  
}
```

REST.Android

- El uso de Volley es muy adecuado para el desarrollo de clientes Rest ya que incorpora una petición preparada para enviar y obtener JSON
- Para hacer peticiones GET:

```
final String url = "http://httpbin.org/get?param1=hello";

// prepare the Request
JsonObjectRequest getRequest = new JsonObjectRequest(Request.Method.GET, url, null,
    new Response.Listener<JSONObject>()
{
    @Override
    public void onResponse(JSONObject response) {
        // display response
        Log.d("Response", response.toString());
    }
},
new Response.ErrorListener()
{
    @Override
    public void onErrorResponse(VolleyError error) {
        Log.d("Error.Response", error.toString());
    }
);

// add it to the RequestQueue
queue.add(getRequest);
```

REST.Android

- En las peticiones POST, para añadir las parejas de parámetros/valor, se reescribe el método `getParams()` y se devuelve un objeto tipo Map

```
url = "http://httpbin.org/post";
StringRequest postRequest = new StringRequest(Request.Method.POST, url,
    new Response.Listener<String>()
{
    @Override
    public void onResponse(String response) {
        // response
        Log.d("Response", response);
    }
},
new Response.ErrorListener()
{
    @Override
    public void onErrorResponse(VolleyError error) {
        // error
        Log.d("Error.Response", response);
    }
}
) {
    @Override
    protected Map<String, String> getParams()
    {
        Map<String, String> params = new HashMap<String, String>();
        params.put("name", "Alif");
        params.put("domain", "http://itsalif.info");

        return params;
    }
};
queue.add(postRequest);
```

REST.Android

- Las peticiones PUT son similares a las POST

```
url = "http://httpbin.org/put";
StringRequest putRequest = new StringRequest(Request.Method.PUT, url,
    new Response.Listener<String>()
{
    @Override
    public void onResponse(String response) {
        // response
        Log.d("Response", response);
    }
},
new Response.ErrorListener()
{
    @Override
    public void onErrorResponse(VolleyError error) {
        // error
        Log.d("Error.Response", response);
    }
}
);

@Override
protected Map<String, String> getParams()
{
    Map<String, String> params = new HashMap<String, String> ();
    params.put("name", "Alif");
    params.put("domain", "http://itsalif.info");

    return params;
}

queue.add(putRequest);
```

REST.Android

- Y peticiones DELETE

```
url = "http://httpbin.org/delete";
StringRequest dr = new StringRequest(Request.Method.DELETE, url,
    new Response.Listener<String>()
{
    @Override
    public void onResponse(String response) {
        // response
        Toast.makeText(getApplicationContext(), response, Toast.LENGTH_LONG).show();
    }
},
new Response.ErrorListener()
{
    @Override
    public void onErrorResponse(VolleyError error) {
        // error.
    }
});
queue.add(dr);
```