Práctica 4:
Acceso a Servicios Web desde el móvil

Introducción y Objetivos

El objetivo de la práctica es el acceso a Servicios Web desde un dispositivo móvil Android, teniendo en cuenta tanto las particularidades de la Web como las de los dispositivos móviles. En concreto, se propone la creación de un cliente que pueda descargar y mostrar fotos del popular sitio Flickr. Este dominio de aplicación se propone sin perjuicio de que los alumnos puedan proponer aplicaciones alternativas de su interés personal, pero teniendo en cuenta que prácticas posteriores extenderán esta con la adición de servicios basados en la localización (LBS).

La app desarrollada debe tener al menos dos actividades: una actividad inicial que permita iniciar una búsqueda y que muestre todos los resultados (por ejemplo, un listado de las URLs o iconos de las diferentes fotos) y una segunda actividad donde se muestren resultados detallados del resultado seleccionado (por ejemplo, la imagen, su título y los hashtags). Se debe gestionar adecuadamente el ciclo de vida de las transiciones; por ejemplo, al regresar a la actividad principal se deben volver a mostrar los resultados de la consulta sin necesidad de enviar una nueva petición a Flickr. Se usaráa la clase `AsyncTask` para la creación de una hebra secundaria donde realizar las tareas de conexión.

El formato y la disposición de los resultados de la búsqueda y del detalle quedan a elección personal. La interfaz inicial podría contener únicamente un botón para comenzar la búsqueda o complicarse un poco más por ejemplo para permitir búsquedas por palabras clave a través de un campo de texto.

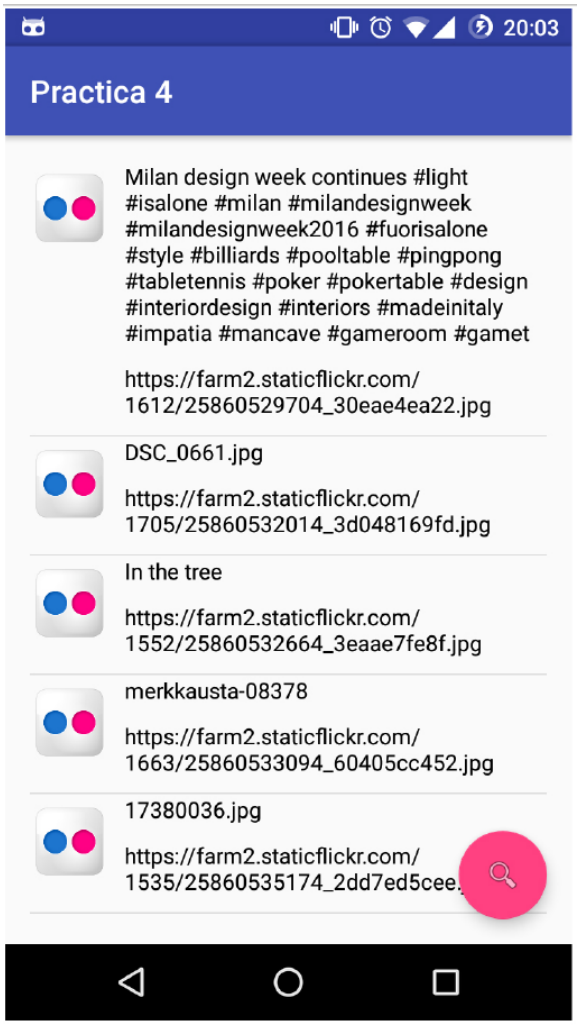
Proveedores de Servicios Web

Actualmente, una gran cantidad de sitios (Facebook, Flickr, Twitter, etc.) ofrecen APIs para acceder a servicios web, junto con sus correspondientes documentaciones. En concreto, recientemente se han popularizado los servicios web basados en REST (*REpresentational State Transfer*) frente a los servicios web “tradicionales” que utilizan SOAP + WDSL + UDDI.

Generalmente, la documentación de estos servicios web incluye los formatos de solicitud y los formatos de respuesta. Merece la pena comentar que los sistemas basados en REST (también llamados *RESTful systems*) utilizan -casi siempre- HTTP como protocolo de comunicación, y utilizan los métodos usuales de HTTP para hacer solicitudes o *requests* (GET, POST…). Sin embargo, REST no tiene un estándar oficial asociado, por lo que diferentes APIs pueden tener diferentes formas de implementar los métodos http. Tampoco está especificado un formato de respuesta, que puede ser XML, JSON, YAML u otros, según la API. Nosotros, en Flickr, utilizaremos REST para las solicitudes, y JSON como formato de respuesta.

Establecimiento de conexiones HTTP en Android

Para establecer una conexión HTTP en Android basta con utilizar las clases `URL` y `HttpUrlConnection`. La primera permite crear un objeto que es una URL a partir de un `String`. La segunda permite crear una conexión HTTP asociada a una URL y proporciona métodos para solicitudes, para obtener la respuesta vía *streams*, o para chequear los códigos de respuesta.



Ejemplo de posible interfaz de la app

Además de estas dos clases, de uso sencillo, hay dos aspectos más a tener en cuenta en Android: las hebras y los permisos de las aplicaciones.

Android no permite conexiones de red en la hebra principal, debido a los posibles retrasos provocados por las conexiones (se lanza una `NetworkOnMainThreadException`). La hebra principal es un bucle que se ejecuta constantemente y se limita a esperar a eventos (como pulsaciones de botones) y responder a los mismos (por ejemplo, con cambios en el interfaz de usuario). Así pues, para gestionar las conexiones a la red habrá que crear una hebra secundaria que se ejecute en un segundo plano.

Una forma de hacer esto es crear una subclase de la clase `AsyncTask` e implementar los métodos `doInBackground()` y `onPostExecute()`. Consulta la documentación de la clase (http://developer.android.com/reference/android/os/AsyncTask.html), que incluye los pasos por los que pasa una tarea asíncrona, incluyendo cómo y para qué son invocados esos dos métodos (sección *"The 4 steps"*), así como las directrices de la sección *"Threading rules"*. Al crear una instancia de dicha clase y llamar a `execute()` se creará una hebra secundaria.

Para conectarse a Internet, la app requiere un permiso especial que el usuario aceptará o rechazará durante la instalación; para solicitarlo, hay que añadir al fichero `AndroidManifest.xml`:

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

Acceso a servicios web: REST y JSON

Solicitudes REST. En la URL http://www.flickr.com/services/api puede consultarse la documentación de la API de Flickr, que permite invocar métodos como `getRecent` o `getContext`, proveyendo los parámetros necesarios (como la API key que se necesita para los dos mencionados, o la ID de la foto para `getContext`), y especificando el formato de la respuesta. Para esta práctica basta con utilizar `getRecent`, que permite obtener las últimas fotos públicas subidas a Flickr, pero podéis utilizar otros métodos adicionales en vuestra app. Así, una solicitud GET a través usará una URL de la forma http://api.flickr.com/services/rest/?method=flickr.photos.getRecent&api_key=xxx&format=json, donde `xxx` es la API key obtenida. Para obtener una API key (no comercial) se necesita crear una cuenta gratuita en Yahoo y acceder a http://www.flickr.com/services/api/keys. Podéis encontrar otros posibles parámetros (e.g., `nojsoncallback`) en: http://www.flickr.com/services/api/response.json.html.

Gestión de las respuestas. Utilizando un stream para la respuesta de la solicitud se puede obtener un `String` con la respuesta en formato JSON como la que devolvería una invocación como la mostrada anteriormente. Para parsear esa respuesta en formato JSON se puede utilizar el paquete `org.json` que incluye Android. Si es necesario, podéis familiarizaros con las clases `JSONObject` y `JSONArray` consultando su documentación. Dado una respuesta en formato JSON, cada JSONObject es un conjunto de parejas nombre-valor encapsulado entre llaves, y cada JSONArray es una lista de JSONObjects separados por comas y encapsulados entre corchetes.

Una vez extraída la información del JSON, habrá que crear una nueva clase para almacenar la información (como mínimo, cada objeto debe tener una URL y un identificador. Una lista (`List`) de dichos objetos almacenará la respuesta completa.

Actualización de la interfaz de usuario

Una vez obtenida la información (de Flickr, o de cualquier otro servicio web), debemos mostrarla de forma apropiada al usuario. Esto implica cambios en la interfaz de usuario y, por tanto, se realizará en la hebra principal, por lo que debe haber algún mecanismo de comunicación entre la hebra secundaria y la principal.

Es común el uso de objetos `Adapter` como interfaz entre los datos y la interfaz gráfica. Tendréis que crear una subclase de `Adapter`, que proveerá de los datos a la interfaz (además de proveer acceso a los datos, el `Adapter` también crea un `View` para cada ítem de los mismos), y utilizar el método `setAdapter()`. Tened en cuenta que dicho método, que es el que realizará los cambios a la interfaz, debe ser llamado siempre que cambia el conjunto de objetos que conforman los datos. Por tanto, se debe invocar dicho método al inicio pero también desde `onPostExecute()`. Este último, recordemos, es un método de la subclase de `AsyncTask` y se ejecuta en la hebra principal y puede por tanto modificar la interfez de usuario. En la documentación de `Adapter` y de `AdapterView` encontraréis más información sobre el uso de estas clases.

Entrega

La entrega se realizará a través de Moodle hasta las 12 horas del **26 de abril** y consistirá en un fichero `.zip` que incluya al menos

- la carpeta `src` del proyecto en Android Studio,
- la aplicación `.apk`, y
- una brevísima documentación en `pdf` indicando los autores y aspectos no triviales.

Ejemplo de hebra principal

```
package com.example.fbobillo.testapplication3;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity
{

  private final static String TAG = "MainActivity";

  private MyAsyncTask myTask = null;


  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    Log.i(TAG, "onCreate");
    setContentView(R.layout.activity_main);

    // Para obtener el valor devuelto en onRetainCustomNonConfigurationInstance
    myTask = (MyAsyncTask) getLastCustomNonConfigurationInstance();

    if (myTask == null) {
      // Evita crear una AsyncTask cada vez que, por ejemplo, hay una rotación
      Log.i(TAG, "onCreate: About to create MyAsyncTask");
      myTask = new MyAsyncTask(this);
      myTask.execute("http://developer.android.com");
    }
    else
      myTask.attach(this);

    Toast.makeText(this, "Hola!", Toast.LENGTH_LONG).show();
  }


  /** Permite devolver un objeto y que persista entre cambios de configuración. Lo
invoca el sistema cuando se va a destruir una actividad y se sabe que se va a
crear otra nueva inmediatamente. Se llama entre onStop y onDestroy. */
  @Override
  public Object onRetainCustomNonConfigurationInstance()
  {
    // Además de devolver mi tarea, elimino la referencia en mActivity
    myTask.detach();
    // Devuelvo mi tarea, para que no se cree de nuevo cada vez
    return myTask;
  }


  public void setupAdapter(Integer integer)
  {
    if (integer != -1)
      Toast.makeText(MainActivity.this,
                     "Codigo de respuesta: " + integer, Toast.LENGTH_LONG).show();
  }

}
```

Ejemplo de tarea asíncrona

```
package com.example.fbobillo.testapplication3;

import android.os.AsyncTask;
import android.util.Log;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;

public class MyAsyncTask extends AsyncTask<String, Void, Integer>
{

  private MainActivity mActivity = null;

  public MyAsyncTask(MainActivity activity)
  {
    attach(activity);
  }


  @Override
  protected Integer doInBackground(String... params)
  {
    HttpURLConnection connection;
    try {
      connection = (HttpURLConnection) new URL(params[0]).openConnection();
      return connection.getResponseCode();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return -1;
  }


  @Override
  protected void onPostExecute(Integer integer)
  {
    if (mActivity == null)
      Log.i("MyAsyncTask", "Me salto onPostExecute() -- no hay nueva activity");
    else
      mActivity.setupAdapter(integer);
  }


  void detach()
  {
    this.mActivity = null;
  }


  void attach(MainActivity activity)
  {
    this.mActivity = activity;
  }

}
```