JSON y servicios web con Android

Práctica 3 de laboratorio

Aplicaciones Telemáticas
ETSIT-UPV

J. E. López, F. J. Martínez Zaldívar

Índice

1. Introducción y objetivos

Los servicios web son utilizados hoy en día en gran cantidad de aplicaciones. En la presente práctica nos planteamos como objetivos ser capaces de, empleando el entorno de programación de Android, utilizar estos servicios elaborando peticiones de tipo API REST, recibir los resultados y extraer de los mismos la información en la que estemos interesados.

Para ello, utilizaremos un ejemplo de entre los múltiples que podemos encontrar en la red: el servicio OpenData de AEMET, la Agencia Estatal de Meteorología. Pediremos la predicción meteorológica para ciertos intervalos o instantes horarios de un municipio en concreto y extraeremos ciertas propiedades de dicha predicción, escribiéndolas en pantalla.

Esta práctica está basada en una anterior en al que se realizaba estos mismos procesos pero en el entorno de Java ejecutado en un PC. Veremos que aunque la mecánica es la misma, existen matices que van a diferenciar sutilmente una práctica de otra.

Los denominadores comunes son:

- Utilización de servicios web: creación de petición API-REST y recolección de resultados.
- Empleo de certificados digitales
- Manipulación de objetos y arrays JSON

Pero como veremos, deberemos tener en cuenta algunas peculiaridades de Java en el contexto Android.

1.1. Formación de grupos

Los grupos para realizar esta práctica serán de uno o dos alumnos, intentando mantener los de la práctica anterior. Si por algún motivo hubiera algún cambio, este debe comunicarse al profesorado. Se sugiere que se lean detenidamente todas las indicaciones dadas en la presente memoria para poder llevar hasta el final la realización de esta práctica.

1.2. Hardware—software necesario y comunicaciones

El hardware y software necesario consistirá en:

- Ordenador personal de cualquier arquitectura: Windows, OS-X o Linux
- IDE Android Studio

- Capacidad de virtualización de procesadores activada o terminal Android real

Respecto a comunicaciones será necesario tener evidentemente conexión a Internet.

2. Servicio AEMET OpenData

Aquí hacemos referencia a la sección "Servicio AEMET OpenData" de la práctica 2 de AA. TT., siendo completamente válidos los aspectos allí comentados.

En el contexto de Android no tenemos disponible la clase `OkHttpClient` para acceder a los servicios web de AEMET, luego tendremos que emplear alguna clase alternativa con la que consigamos los mismos objetivos. Una alternativa es `HttpURLConnection`. Un pequeño ejemplo de uso de esta clase se muestra a continuación en forma de método o función que, a partir de una URL introducida como un parámetro de la clase `String` en la llamada, devuelve un `String` con el resultado de la petición API-REST. Se sugiere que se entienda perfectamente el funcionamiento del siguiente código o *snippet* y que se utilice oportunamente en la solución.

Llamada a servicio web

```
/** La peticion del argumento es recogida y devuelta por el método API_REST.
    Si hay algun problema se retorna null */
public String API_REST(String uri) {

    StringBuffer response = null;

    try  {
        url = new URL(uri);
        Log.d(TAG, "URL: " + uri);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        // Detalles de HTTP
        conn.setReadTimeout(15000);
        conn.setConnectTimeout(15000);
        conn.setRequestMethod("GET");

        int responseCode = conn.getResponseCode();
        Log.d(TAG, "Codigo de respuesta: " + responseCode);
        if (responseCode == HttpsURLConnection.HTTP_OK) {
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(conn.getInputStream()));
            String output;
            response = new StringBuffer();

            while ((output = in.readLine()) != null) {
                response.append(output);
            }
            in.close();
        } else {
            Log.d(TAG, "responseCode: " + responseCode);
            return null; // retorna null anticipadamente si hay algun problema
```

```
            }
        } catch(Exception e) {
            // Posibles excepciones: MalformedURLException, IOException y ProtocolException
            e.printStackTrace();
            Log.d(TAG, "Error conexión HTTP:" + e.toString());
            return null; // retorna null anticipadamente si hay algun problema
        }

        return new String(response); // de StringBuffer -response- pasamos a String

    } // API_REST
```

A partir de aquí, habría que realizar el oportuno análisis JSON y extraer las propiedades en las que estamos interesados. Recuérdese de la práctica de AEMET realizada con Java en un PC, que habrá que efectuar dos llamadas encdadenadas para obtener el JSON final.

3. JSON y el paquete `org.json` de Java

De la misma forma que ocurría con el paquete `okhttp3`, el paquete `javax.json` no lo vamos a encontrar en Android. Alternativamente, el paquete de Android `org.json` puede cumplir perfectamente su cometido, teniendo una funcionalidad similar, aunque con algunas diferencias, a la del citado paquete `javax.json`.

Por ejemplo, podemos crear directamente un array u objeto JSON a partir del string que lo representa con:

```
JSONArray array = new JSONArray(string_JSON_Array);
o bien
JSONObject objeto = new JSONObject(string_JSON_Object);
```

A partir de un `JSONArray` o `JSONObject`, pueden extraerse objetos, arrays, strings, enteros, booleanos, etc.:

- Objetos: `.getJSONObject(selector)`
- Arrays: `.getJSONArray(selector)`
- Enteros: `.getInt(selector)`
- Strings: `.getString(selector)`
- Booleanos: `.getBoolean(selector)`
- ...

donde `selector` hace referencia a un entero, si estamos accediendo a un elemento de un array o a un string si estamos accediendo a una propiedad de un objeto.

4. Instalación de certificados digitales

Tal y como ocurría con la práctica de AEMET con Java en un PC, será necesario que el certificado digital (o la cadena oportuna de certificados digitales) esté instalado en algún tipo de almacén al que acuda el sistema cuando se pretenda acceder a un recurso web mediante el protocolo `https`, lo cual es el caso. En la práctica anterior, descargamos y formamos este almacén. En esta, muy probablemente tendremos ya instalado de manera correcta el certificado necesario en el almacén propio de Android, por lo que no habrá que hacer nada.

Para verificar si efectivamente el certificado está instalado o no, tanto si se va a utilizar el terminal virtual de Android, como si el terminal es real, el certificado debería encontrarse en: Configuración del sistema (Settings) → Seguridad y ubicación (Security and Location) → (Advanced) → Cifrado y Credenciales (Encryption and credentials) → Certificados de confianza (Trusted credentials) → y definitivamente, deberíamos poder encontrar el certificado ACCVRAIZ1 de ACCV (incluso deberíamos poder activarlo o desactivarlo por si pudiera estar comprometido, o por si por algún motivo no deseáramos considerarlo).

Si efectivamente, así lo encontramos, lo tenemos ya perfectamente preparado para que todo funcione; si no lo encontramos, deberemos ir a la página web de ACCV (`https://www.accv.es`, tal y como se hizo en la práctica 2) e instalar, esta vez, de manera directa el certificado en nuestro terminal móvil —sin necesidad de crear ningún almacén—, siguiendo las indicaciones que se muestran al clicar sobre el certificado descargado en nuestro terminal Android.

5. Clase `AsyncTask`

Hay un problema añadido que todavía no hemos considerado: la petición HTTP(S) no debemos ponerla en el mismo hilo de ejecución que el hilo en el que se está ejecutando la interfaz gráfica con el usuario, ya que el tiempo de respuesta del servidor puede ser relativamente largo, y si tenemos bloqueada la ejecución de dicho hilo, mientras tanto, el móvil no podría atender a ninguna interacción que el usuario hiciera, por lo que aparecerá como bloqueado. Ello implica que será necesario realizar dicha petición HTTP en un hilo distinto al que se ejecuta la UI (User Interface). Podemos emplear hilos utilizando la clase `Thread`, pero Android nos proporciona una clase con la que se simplifica notablemente todo ello; se denomina `AsyncTask`. Esta es una clase abstracta a la que le queda pendiente solo un método por implementar

(abstracto): en este método incluiremos ese código que puede requerir mucho tiempo de ejecución o puede bloquear la interacción con el terminal, con lo que habremos resuelto ese problema.

Un funcionamiento resumido y simplificado de esta clase `AsyncTask` es el que aparece en la figura 1.

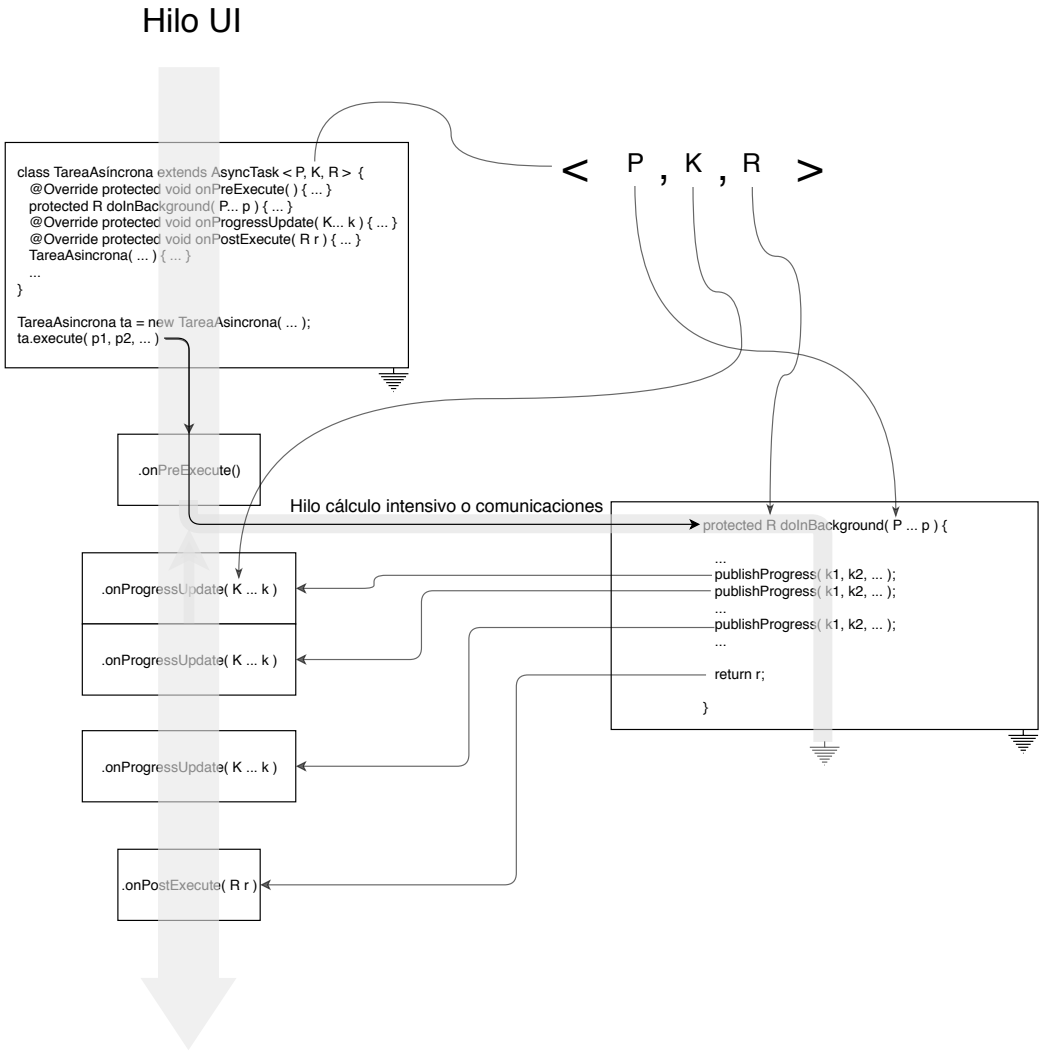

Figura 1: Tarea asíncrona

Un resumen de los detalles más importantes es el siguiente:

- Hay que extender la clase AsyncTask, parametrizándola con las clases genéricas `P`, `K` y `R`, cuyo siginificado se explicará a continuación.
- Hay que escribir el método `protected R doInBackground(P...p)` admitiendo un número indeterminado de parámetros de la clase `P` (de ahí la notación con la elipsis ...) y devolviendo un objeto de la clase `R`.
- La sobreescritura de los restantes métodos es opcional. Algunos de ellos tendrían la siguiente funcionalidad (todos ellos, a diferencia de `doInBackground` se ejecutan ya en el hilo de la UI, por lo que pueden interactuar directamente con esta):
  - `@Override protected void onPreExecute()`: código a ejecutar antes de abrir el hilo paralelo
  - `@Override protected void onPostExecute(R r)`: código a ejecutar tras finalizar el hilo paralelo (es decir, tras finalizar el método `doInBackground`). El parámetro que recibe (de la clase `R`) es precisamente el valor devuelto por `doInBackground`
  - `@Override protected void onProgressUpdate(K...k)`: se produce su invocación, si se produce la llamada a `publishProgress(k1, k2, ...)` dentro del método `doInBackground`. Se emplea para ir *informando* del progreso del hilo alternativo en el hilo de la UI, ya que sólo se puede actualizar gráficamente la UI desde el propio hilo de la UI y no desde otro.
- Como en cualquier clase de Java, el constructor es completamente opcional.
- Una vez definida la clase, se crea un objeto de la misma (`ta` en la figura) empleando el constructor oportuno y a continuación se ejecuta el método `execute` (en el ejemplo, `ta.execute(p1, p2, ...)`). Obsérvese que pueden emplearse cualquier cantidad de parámetros de la clase `P` en la llamada a `execute`. Recuérdese que para acceder a cada uno de estos parámetros (dentro del método `doInBackground(P...p)`), simplemente tendremos que acceder a `p[0]`, `p[1]`, hasta `p[p.length-1]`.

6. Permisos

Si el código estuviera ya finalizado y en este instante se ejecutara, muy probablemente no funcionaría a no ser que hubieramos tenido en cuenta

un detalle importante: hay que acceder a Internet y para ello la aplicación *debe pedir permiso* para ello. Para solucionar este problema se sugiere que se busque la solución en la documentación de Android; para ello, se tendrá que modificar el fichero `AndroidManifest.xml` añadiendo oportunamente cierto elemento XML.

7. Tratamiento de errores

Las principales fuentes de error que podemos encontrarnos en nuestra aplicación son, por una parte, por un código de respuesta del protocolo HTTP que diera lugar a un error, lo cual podría controlarse en la sentencia `else` de la rutina anterior `API_REST`:

Error en protocolo HTTP(S)

```
if (responseCode == HttpsURLConnection.HTTP_OK) {
   ...
} else {
   Log.d(TAG, "responseCode: " + responseCode);
   return null; // retorna null anticipadamente si hay algun problema
}
```

por otra, por alguna excepción como `MalformedURLException`, `IOException` y `ProtocolException`, lo cual es controlable en el bloque `catch` del mismo método:

```
   } catch(Exception e) {
      // Posibles excepciones: MalformedURLException, IOException y ProtocolException
      e.printStackTrace();
      Log.d(TAG, "Error conexión HTTP:" + e.toString());
      return null; // retorna null anticipadamente si hay algun problema
  }
```

En cualquier caso, el denominador común que se ha adoptado es que la rutina finaliza y retorna un `null`.

Si ocurre esta situación, la intención es generar un `Toast` o pequeña advertencia o mensaje temporal que suele aparecer en la parte inferior de la pantalla. Puede conseguirse con la siguiente instrucción:

Toast

```
Toast.makeText( getApplicationContext(), "Problemas en el servicio web de AEMET",
                Toast.LENGTH_LONG ).show();
```

Ya que está íntimamente relacionado con la UI, esta llamada se debe efectuar en el hilo de la UI, por ejemplo, podría ejecutarse en `protected void onPostExecute(String respuesta)` —cuando el *string* `respuesta` fuera `null`—

Por último, también puede ocurrir una excepción al decodificar la respuesta JSON con la excepción `JSONException`, debiendo *cazarla* allá donde sea oportuno.

8. Acceso a Internet

Si se está empleando un terminal virtual o AVD para llevar a cabo la práctica, será necesario comprobar si es posible acceder a Internet desde el mismo, para lo cual se sugiere que se abra el navegador que trae el propio terminal virtual y se abra una página web conocida. Si no hay problemas, puede suponerse que tenemos acceso a Internet y hay que hacer caso omiso a lo que resta de sección.

Si no es posible realizar la carga de estas páginas, existe la posibilidad de que el problema sea el DNS que está utilizando el terminal virtual.

Si esta es la causa del problema, ello posiblemente pueda solucionarse de la siguiente manera: primeramente hay que identificar el nombre del AVD y después reajustar su DNS. En Windows habría que operar de la siguiente manera (si la instalación de Android Studio se ha realizado de manera convencional):

Sólo si no hay acceso a Internet desde AVD (Windows)

```
// Listar los AVD que tenemos en nuestro IDE:
> c:\Users\%Username%\AppData\Local\Android\sdk\emulator\emulator -list-avds
Nexus_5X_API_21
Nexus_5X_API_24_no_Google_API_
Nexus_5X_API_25
...

// Hay que identificarlo (supongamos que es el primero) y ajustar su DNS:
> c:\Users\%Username%\AppData\Local\Android\sdk\emulator\emulator @Nexus_5X_API_21 -dns-server 8.8.8.8
// Probablemente nos aparezca este error al que haremos caso omiso
emulator: ERROR: Running multiple emulators with the same AVD is an experimental feature.
Please use -read-only flag to enable this feature.
```

En Mac-OSX (similar en sistemas Linux):

Sólo si no hay acceso a Internet desde AVD (Mac-OSX)

```
// Listar los AVD que tenemos en nuestro IDE:
$ ~/Library/Android/sdk/emulator/emulator -list-avds
Nexus_5X_API_21
Nexus_5X_API_24_no_Google_API_
Nexus_5X_API_25
...

// Hay que identificarlo (supongamos que es el primero) y ajustar su DNS:
$ ~/Library/Android/sdk/emulator/emulator  @Nexus_5X_API_21 -dns-server 8.8.8.8
// Probablemente nos aparezca este error al que haremos caso omiso
emulator: ERROR: Running multiple emulators with the same AVD is an experimental feature.
Please use -read-only flag to enable this feature.
```

9. Desarrollo de la práctica

Un resumen de las acciones a realizar puede ser el siguiente:

- Clónese el repositorio desde `Android Studio`.
- En el directorio clonado aparecerá un fichero denominado `README.md` con indicaciones iniciales para comenzar la realización de la práctica, así como un directorio denominado `doc` donde reside la memoria de la práctica.
- Recuérdese los detalles del funcionamiento de las API de AEMET OpenData
- Verifíquese la instalación del certificado raíz de ACCV
- Complétese el fichero `MainActivity.java` del proyecto Android, siguiendo las indicaciones que aparecen en el propio fichero y las dadas en la presente memoria
- Complétese el fichero de recursos `activity_main.xml` del proyecto Android con los elementos `View` oportunos para indicar las magnitudes y los valores meteorológicos exigidos:
- Gestiónese el permiso de acceso a Internet oportunamente
- Obténgase la predicción meteorológica siguiente:
  - Temperatura (a las 18 h —intervalo 12–18 h—)
  - Probabilidad de precipitación en tanto por cien entre las 12 y las 18 h
  - Dirección y velocidad del viento entre las 12 y las 18 h
  - Estado del cielo entre las 12 y las 18 h

  del día siguiente al que se realice la práctica, en alguna de las siguientes poblaciones de España:

  - Peleas de Abajo, provincia de Zamora.
  - Guarromán, provincia de Jaén
  - Malcocinado, provincia de Badajoz

  Deben contrastarse los resultados con los obtenidos visualmente de manera directa de la página web de AEMET.

10. Resultados a entregar

Al finalizar la fecha tope prevista para la realización de la práctica, debe actualizarse el repositorio en GitHub, indicando mediante un *tag* la denominación **pr3.sesion.fin** (recuérdese subir los *tags* explícitamente, ya que no es suficiente con actualizar el repositorio remoto con el local). Insistimos en ello para que no se confundan los nombres de las `tags`, con los comentarios de los *commits* ya que son conceptos distintos. Si se proporciona tiempo adicional para finalizar los detalles que pudieran quedar pendinetes, las sucesivas versiones operativas que se suban a GitHub deben estar etiquetadas con **pr3.1.fin**, **pr3.2.fin**, etc.

Se sugiere que se vayan realizando *commits* para ir guardando en el respositorio local las instantáneas del directorio de trabajo que os parezcan oportunas, así como realizar una sincronización del repositorio local con el remoto en GitHub, para salvaguardar una copia de seguridad del repositorio en la nube.

Si realizáis consultas através de las *issues* de GitHub, recordad antes actualizar el repositorio remoto en GitHub e indicar como *assignees* tanto al profesor como a todos los integrantes del grupo para que lleguen notificaciones vía email de que hay una *issue* pendiente de ser contestada.

Se tomará como referencia del trabajo definitivo, el *commit* de la rama `master` con el *tag* de la forma `pr3.n.fin` o en su defecto, `pr3.sesion.fin`.

La ampliación de esta práctica podría considerarse perfectamente como **trabajo de asignatura**. Entiéndase por ampliación, la posibilidad de, por ejemplo, seleccionar todos los pueblos de cierta región o incluso de cualquier región, o plantear otros casos de servicios dados por AEMET, mejora de la interfaz gráfica añadiendo figuras para representar el estado del cielo, la temperatura, etc., o cualquier concepto que el alumno desee desarrollar.