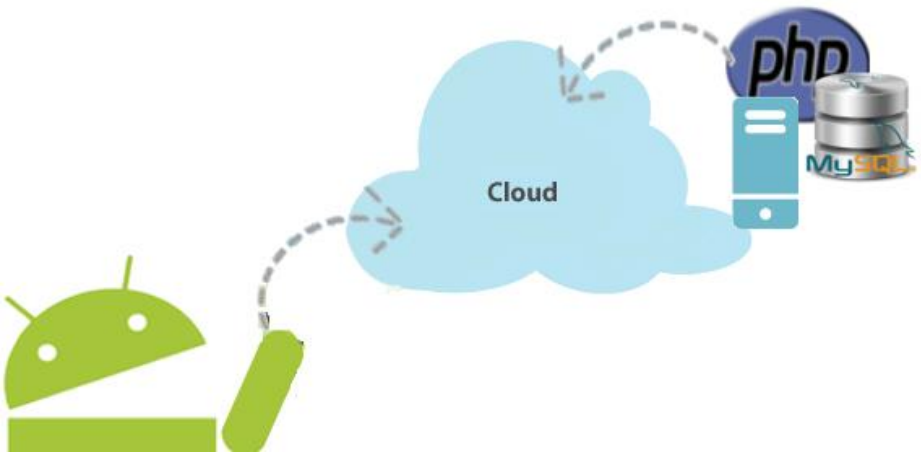
Acceso a datos remotos



Contenido

Acceso a datos remotos

Supongamos que deseamos que una aplicación haga uso de una BD (MySQL, Oracle, SQLServer,…) situada en un servidor externo al dispositivo.
Puede programarse una aplicación Android con acceso a la BD de varias formas distintas:

- **Directamente mediante programación Java**:
  - Necesita disponer del driver JDBC adecuado para el SGBD
  - Dejar usar así el servidor presenta problemas de seguridad (clave de acceso al servidor es relativamente "hackeable", son fáciles las inyecciones SQL, …)
  - Por tanto, solo debe utilizarse en entornos seguros (ej.: acceso a una BD de una intranet de una empresa desde dispositivos que solo se conectan en el trabajo,…) o con código bien ofuscado.
- **Utilizando servicios WEB**:
  - Necesita que el servidor ofrezca el servicio ya sea SOAP, REST,…
  - Más complejo de programar
  - Si el servicio es público (Twitter, Facebook, etc.) hay que conocer la respuesta que ofrece.
  - El servidor sirve los datos en formato XML, JSON,… Tendrán que ser "parseados" (el fichero se analiza para obtener la información relevante, sin etiquetas, marcas, etc.)

Probando con servidor en localhost y emuladores

- IP's en el emulador incluido en el SDK:

Network Address	Description
10.0.2.1	Router/gateway address
10.0.2.2	Special alias to your host loopback interface (i.e., 127.0.0.1 on your development machine)
10.0.2.3	First DNS server
10.0.2.4 / 10.0.2.5 / 10.0.2.6	Optional second, third and fourth DNS server (if any)
10.0.2.15	The emulated device's own network/ethernet interface
127.0.0.1	The emulated device's own loopback interface

  Si el servidor está en localhost la IP que debemos poner en nuestro proyecto para trabajar desde el emulador es 10.0.2.2.
- Si utilizas el emulador Geanymotion, las ip's no son 10.0.**2**.etc, son 10.0.**3**.etc.

Tarea costosa

Desde Android v3.0 obliga a hacer las operaciones de acceso a red en un **hilo independiente**, por tanto debemos elegir una de las formas habituales de conseguirlo: hilos o tareas asíncronas (recuerda que estas facilitan trabajar con la interfaz de usuario) o **Volley**.

Permisos

Siempre es necesario:

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

Comprobación de la disponibilidad de la red

Obviamente la red en un dispositivo Android no siempre está disponible. Para comprobar la disponibilidad, podemos programar un método al que llamaremos antes de realizar la conexión:

```
public boolean isNetworkAvailable() {
    ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo redActiva = cm.getActiveNetworkInfo();
    return redActiva != null && redActiva.isAvailable() && redActiva.isConnected();
}
```

Requiere permiso ACCESS_NETWORK_STATE en AndroidManifest:

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

Si lo que se desea es un determinado tipo de conexión (wifi o móvil, etc.), podemos usar activeNetwork.getType(); (como ya vimos en el tema BroadcastReceiver).

Conexión directa a BD MySQL

Implementando la BD en el servidor

En nuestro servidor MySQL se ha creado la BD bdpdm y el usario de nombre pdm con password contra_pdm que puede acceder al servidor y que solo tiene privilegios para trabajar con dicha BD:

```
CREATE DATABASE bdpdm;
USE bdpdm;

CREATE TABLE tablapdm (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(30) NOT NULL
);

INSERT INTO tablapdm VALUES (1,'uno');
INSERT INTO tablapdm VALUES (2,'dos');

CREATE USER 'pdm'@'localhost' IDENTIFIED BY 'contra_pdm';
GRANT USAGE ON *.* TO 'pdm';
GRANT ALL PRIVILEGES ON bdpdm.* TO 'pdm';
FLUSH PRIVILEGES;
```

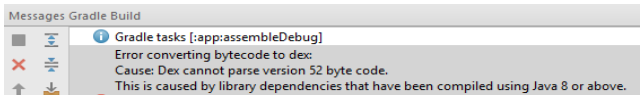
Proyecto P_86_AccesoMySQL

El proyecto es tan simple que se conecta a la BD bdpdm para mostrar el nombre del primer registro de la tabla tablapdm.

Controlador JDBC

Necesitamos la librería para poder trabajar desde Java con SGBD MySQL.

Hay que añadir la dependencia a la librería mysql-connector 5.1.36 (es la última compatible con Android, las siguientes están compiladas con java8 y Android todavía no admite todas las funciones de java8).



Tal y como indica el enlace añadiremos la dependencia en el fichero buil.gradle:

```
implementation group: 'mysql', name: 'mysql-connector-java', version: '5.1.36'
```

Ya podremos programar en Java conexiones a MySQL (documentación).

(Si el servidor es MySQL 8, necesita driver 8 !!!)

Clase MiRegistro (tipo POJO)

```
public class MiRegistro {
    int id;
    String nombre;

    public Registro(int id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Clase MainActivity

Solo va a consultar la tabla para obtener el nombre del primer registro:

```
public class MainActivity extends AppCompatActivity {

    private TextView textView;
    private List<MiRegistro> listaRegistros;

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

        Button bt= findViewById(R.id.button);
        textView = findViewById(R.id.textView);
        bt.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ConnectivityManager connMgr = (ConnectivityManager)
                        getSystemService(Context.CONNECTIVITY_SERVICE);
                NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
                if (networkInfo != null && networkInfo.isConnected()) {
                    textView.setText(R.string.espere);
                    new Tarea().execute();
                } else {
                    textView.setText(R.string.imposible);
                }
            }
        });
    }

    private class Tarea extends AsyncTask<Void, Void, List<MiRegistro>> {
        @Override
        protected List<MiRegistro> doInBackground(Void... params) {
            listaRegistros = BDConectar.getDatosRegistro();
            return listaRegistros;
        }

        @Override
        protected void onPostExecute(List<MiRegistro> result) {
            if (listaRegistros !=null)
                textView.setText(listaRegistros.get(0).getNombre());
            else
                textView.setText(R.string.sindatos );
        }
    }

}
```

Clase BDConectar

Es la que utiliza la librería descargada, contiene los métodos para conectar/desconectar con el servidor y los métodos para CRUD (insertar, leer, actualizar y borrar registros) en la tabla.

```
public class BDConectar {
    private static Connection connection;
    private static Statement statement;
    //Método que crea la conexion connection la BD
    private static void crearConexion() throws InstantiationException, IllegalAccessException, ClassNotFoundException {
        //Cargamos el driver de conexión a la base de datos en MySQL
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        //Para crear la conexión a la base de datos usamos el método getConnection de la clase DriverManager
        //que devuelve un objeto Connection que usaremos para crear y ejecutar sentencias SQL sobre la base de datos.
        try {
            //Datos del servidor y puerto MySQL, del usuario; de la BD, de la tabla y del conector
            String host = "10.0.2.2:3306";
            String usuario = "pdm";
            String password = "contra_pdm";
            String baseDatos = "bdpdm";
            String cadenaConexion = "jdbc:mysql://" + host + "/" + baseDatos;
            connection = (Connection) DriverManager.getConnection(cadenaConexion, usuario, password);
            //El método createStatement() de la clase Connection devuelve un objeto Statement,
            // connection el cual podremos ejecutar sentencias en la BD.
            statement = connection.createStatement();
        } catch (Exception e) {
            Log.e("Error",e.getMessage());
        }
    }

    //Método que cierra la conexion connection la BBDD
    private static void cerrarConexion() {
        try {
            if (statement != null) {
                statement.close();
            }
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            Log.e("Error",e.getMessage());
        }
    }

    //Método que devuelve un ArrayList connection objetos de tipo MiRegistro
    static List<MiRegistro> getDatosRegistro() {
        List<MiRegistro> listaRegistros = new ArrayList<>();
        String sentenciaSQL = "SELECT id, nombre FROM tablapdm";
        try {
            crearConexion();
            //el método executeQuery sirve para ejecutar consultas sobre la base de datos.
            // Recibe como parametro una sentencia SQL y devuelve un objeto ResultSet que contendrá los datos de la consulta.
            //Existe otra forma de consultar, de una manera un poco más elegante, utizando objetos ResultSetMetaData
            ResultSet resultSet = statement.executeQuery(sentenciaSQL);
            //El bucle nos permitirá iterar indefinidamente, mientras en el objeto ResultSet hayan más registros.
            while (resultSet.next()) {
                //Podemos obtener los datos de la fila actual connection los métodos getInt(int) y getString(int) de ResultSet.
                // A dichos métodos le pasamos como parametro el nombre de la columna que queremos obtener.
                //Una vez obtenidos los datos, los añadimos a la lista.
                listaRegistros.add(new MiRegistro(resultSet.getInt("id"), resultSet.getString("nombre")));
            }
            resultSet.close();
        } catch (Exception e) {
            Log.e("Error",e.getMessage());
        } finally {
            cerrarConexion();
        }
        return listaRegistros;
    }

    //Método que actualiza datos de la tabla. Recibe como parámetro un objeto de tipo MiRegistro.
    public static void updateRegistro(MiRegistro miRegistro) {
        String sentenciaSQL;
        try {
            crearConexion();
            //Si el Id está a cero es un nuevo registro
            if (miRegistro.getId() == 0) {
                sentenciaSQL = "INSERT INTO tablapdm (nombre) VALUES (" + miRegistro.getNombre() + ")";
            }
            //Si es distinto de cero es una actualización del registro.
            else {
                sentenciaSQL = "UPDATE tablapdm SET nombre = " + miRegistro.getNombre() + " WHERE id = " + miRegistro.getId();
            }
            //Método para inserción y modificación de datos
            statement.executeUpdate(sentenciaSQL);
        } catch (Exception e) {
            Log.e("Error",e.getMessage());
        } finally {
            cerrarConexion();
        }
    }

    //Método que borra datos de la tabla
    public static void deleteRegistro(MiRegistro miRegistro) {
        String sentenciaSQL;
        sentenciaSQL = "DELETE FROM tablapdm WHERE id = " + miRegistro.getId();
        try {
            crearConexion();
            statement.executeUpdate(sentenciaSQL);
        } catch (Exception e) {
            Log.e("Error",e.getMessage());
        } finally {
            cerrarConexion();
        }
    }

}
```

PROBLEMA!

Acceso a Oracle y SQL Server

Conexión desde android a Oracle
Conectar android a SQL Server 2008
(Son una base sobre la que trabajar ya que no funcionan al usar solo el hilo principal)

Servicios Web

Actualmente los servidores ofrecen Servicios Web que pueden devolver la información de la web en formato HTML (para ser visualizada directamente por navegadores web) y en formato XML o JSON u otro tipo (para ser interpretada y clasificada por una aplicación).
La W3C define "Servicio web" como un **sistema de software diseñado para permitir interoperatibilidad máquina a máquina en una red**. Se trata de API's que son publicadas, localizadas e invocadas a través de la web. Es decir, una vez desarrolladas, son instaladas en un servidor y otras aplicaciones (u otros servicios Web) pueden descubrirlas desde otros ordenadores e invocar uno de sus servicios.
Como norma general, el transporte de los datos se realiza a través del protocolo HTTP y la representación de los datos mediante estándares abiertos (XML/JSON).

(Algunas) Tecnologías de Web Services

Existen dos grandes filosofías a la hora de escribir servicios web:

- **REST**: utiliza XML, JSON y HTTP. Cada URL representa un objeto sobre el que puedes realizar POST, GET, PUT y DELETE (las operaciones típicas del HTTP, comparables a las CRUD "Create, Read, Update, Delete" de los SGBD). Ventajas:
  - Ligero: transferencia de objetos JSON o XML.
  - Resultados legibles.
  - Fácil de implementar: no hacen falta herramientas específicas.
- **SOAP**: es toda una infraestructura basada en XML donde cada objeto puede tener métodos definidos por el programador con los parámetros que sean necesarios. Ventajas:
  - Fácil de consumir
  - Rígido: tipado fuerte, sigue una DTD
  - Herramientas de desarrollo

REST vs. SOAP:
Muchos diseñadores de Servicios Web han llegando a la conclusión que SOAP es demasiado complicado. Por tanto, han comenzando a utilizar Servicios Web basados en REST para mostrar cantidades de datos masivos. Es el caso de grandes empresas como eBay y Google.

Formatos más utilizados en el intercambio de datos

- estándar **XML** (puro o sus "versiones" de RSS, Atom,….): ya es un "viejo" conocido nuestro.
- **JSON** (JavaScript Object Notation) es otro formato ligero para el intercambio de datos.

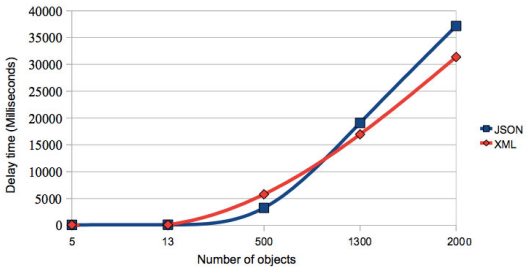
Estudio comparativo

JSON:

```
{"menu": {
"id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

XML:

```
<menu id="file" value="File">  <popup>
<menuitem value="New" onclick="CreateNewDoc()" />
   <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```



Formato JSON

El JSON es una forma de codificar objetos, arrays o cualquier otra serie de datos en un string y posteriormente poder descodificarlo sin muchos problemas.
JSON es un subconjunto de la notación literal de objetos de JavaScript.
Esto es especialmente útil para enviar información entre cliente y servidor de una forma muy sencilla, puesto que cada componente descodifica la información según le convenga, indistintamente del resto del sistema. Originalmente estaba destinado ser usado para Javascript, pero cada vez más lenguajes lo soportan de forma nativa.
Es la misma idea que XML, sin embargo, el JSON está pensado minimizando el uso de caracteres para conseguir una codificación más pequeña.
La simplicidad de JSON ha dado lugar a la generalización de su uso como alternativa a XML. Ver tutorial JSON w3schools.

Elementos JSON

An JSON file consist of many components. Here is the table defining the compoents of an JSON file and their description:

Sr.No	Component & description
1	**Array([])** In a JSON file , square bracket ([) represents a JSON array
2	**Objects({)** In a JSON file, curly bracket ({) represents a JSON object
3	**Key** A JSON object contains a key that is just a string. Pairs of key/value make up a JSON object
4	**Value** Each key has a value that could be string , integer or double e.t.c

Ejemplo de JSON Array

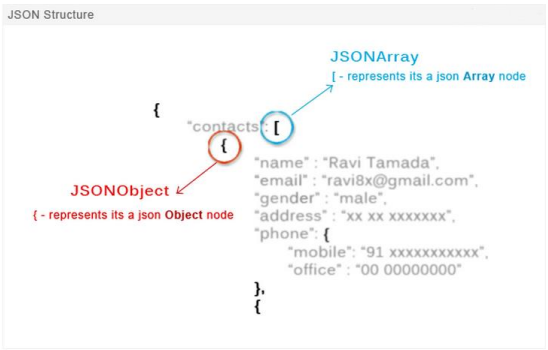
```
["Elemento 1","Elemento 2"]
```

Ejemplo de JSON Object

```
{"Campo 1":"Valor 1" , "Campo 2":"Valor 2" }
```

Ejemplo de JSON Array con JSON Object

```
{"empleados": [
{"nombre":"Juan" , "apellido":"Perez" },
{"nombre":"Ana" , "apellido":"Gomez" },
{"nombre":"Pedro" , "apellido":"Hernandez" }
]}
```



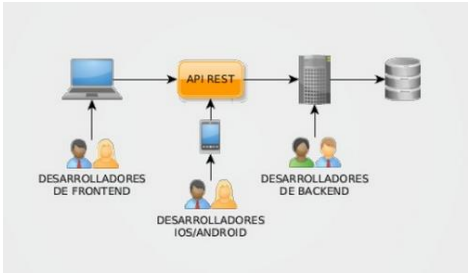
Utilidades para el navegador

Para poder "parsear" el JSON es necesario conocer su modelo, desde el navegador puede ser dificultoso

Ejemplo de JSON	Visor de JSON
	

Servicio Web REST

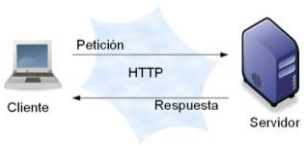
REST (Representational State Transfer) es un **estilo arquitectónico** de Servicio Web orientado a construir aplicaciones distribuidas y basado en las características de la web.



REST funciona sobre HTTP y estructura la información en XML o en JSON.

Un servidor RESTfull (el que implementa servicio REST) tiene todos sus recursos direccionados por URLs accesibles mediante peticiones HTTP.

Una petición puede tener múltiples objetivos dependiendo del método del protocolo HTTP que se elija:



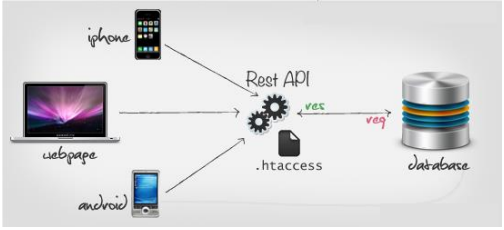
- se usa método **POST** para insertar un recurso
- se usa método **GET** para obtener un recurso (lo que vamos a ver en esta unidad)
- se usa método **PUT** para cambiar el estado de un recurso
- se usa método **DELETE** para eliminar un recurso

Todos los métodos ejecutan una petición (request) y obtienen una respuesta (response).

Es decir, el servidor que implemente REST (los llamados servidores RESTful) devuelve la información solicitada por el cliente mediante una petición GET en formato XML o JSON.

Rest API

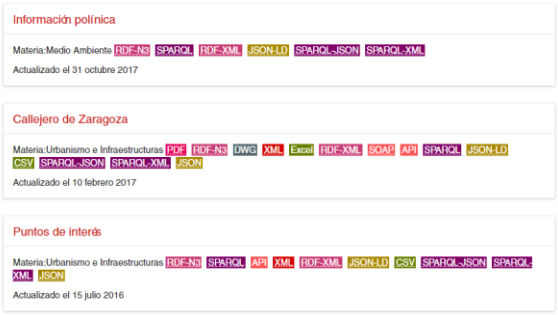
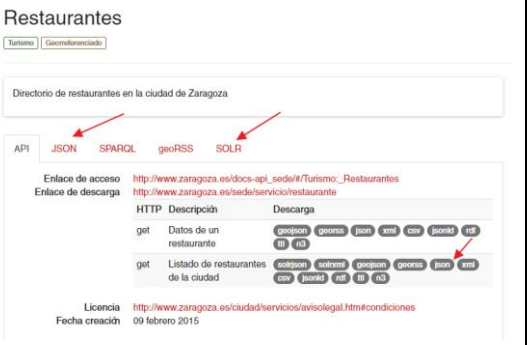
Las famosas *API's* que publican muchos de los sitios web no son más que servicios web de este tipo, aunque en la mayoría de los casos con medidas de seguridad adicionales tales como autenticación OAuth o similares. Para poder utilizarlas hay que leer su documentación (a menudo, es lo más difícil de encontrar!).



¿Quién expone API's?

Ayto. de Zaragoza Aragón Open Data Datos públicos estatales AEMET (ejemplo Zaragoza sin clave) … Tú mismo en un servidor utilizando el lenguaje que prefieras (ver CrearServiciosWeb.pdf guardado en carpeta recursos)	

Datos Abiertos Ayto. Zaragoza

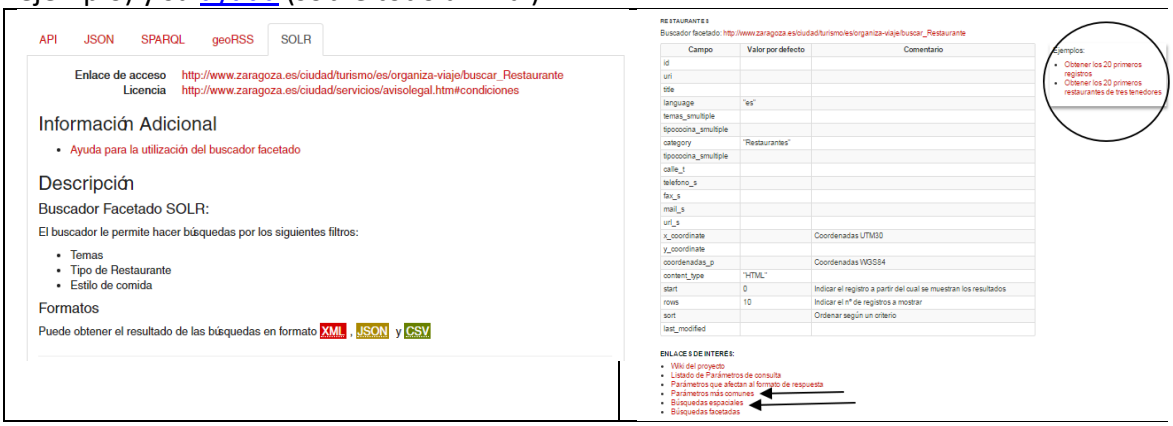
Catálogo 	Ejemplo de uso 

<u>Ejemplo de uso de datos abiertos del Ayto. de Zaragoza:</u>

Supongamos que deseamos una aplicación que devuelva los restaurantes más cercanos a la estación de Renfe de Zaragoza (obtener su geolocalización lo veremos en el siguiente tema):

Alternativa 1:

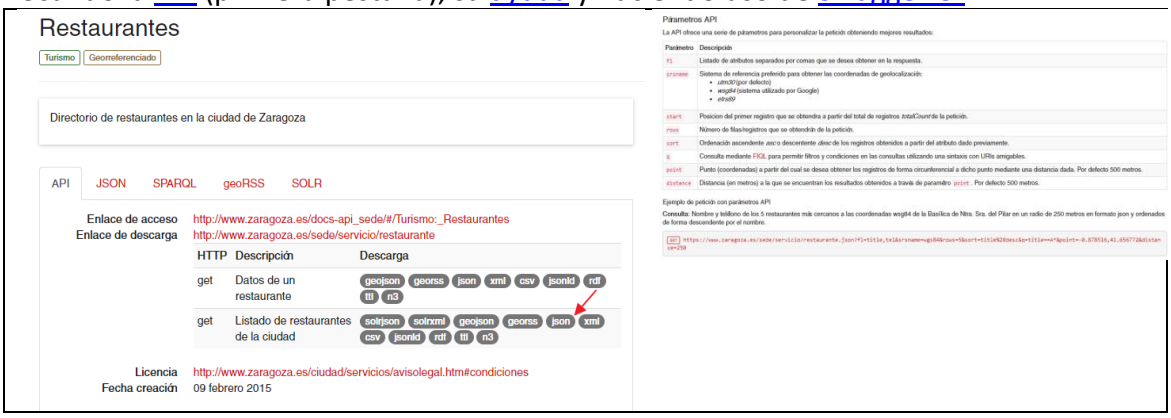
La solución es usar el buscador facetado SOLR (indicado en la última pestaña del ejemplo) y su ayuda (sobre todo al final)



- URL para JSON de todos los restaurantes es:
  http://www.zaragoza.es/buscador/select?wt=json&q=*:*%20AND%20-tipocontenido_s:estatico%20AND%20category:Restaurantes
  El resultado es 1148 restaurantes, aunque parezca solo vemos 50. (Te recuerdo que %20 indica espacio en blanco en la dirección URL)
- Si no has instalado ningún visor JSON en el navegador, para las pruebas puedes añadir que lo tabule (&indent=true), para tu aplicación no es necesario:
  http://www.zaragoza.es/buscador/select?wt=json&indent=true&q=*:*%20AND%20-tipocontenido_s:estatico%20AND%20category:Restaurantes
  El resultado obviamente sigue siendo 1148
- URL que omite la cabecera de la respuesta (&omitHeder=true):
  http://www.zaragoza.es/buscador/select?wt=json&indent=true&q=*:*%20AND%20-tipocontenido_s:estatico%20AND%20category:Restaurantes&omitHeader=true
- URL para JSON que muestra nombre, calle, tfno. y coordenadas de todos los restaurantes (introducimos mediante la clausula fl la lista de los nombres de los campos a mostrar):
  http://www.zaragoza.es/buscador/select?wt=json&indent=true&q=*:*%20AND%20category:Restaurantes&omitHeader=true&fl=title,calle_t,telefono_s,coordenadas_p
- URL para JSON que muestra nombre, calle, tfno. y coordenadas de los restaurantes que están en un radio de 2Km de la estación de trenes que tiene coord: 41.658648,-0.909592 (introducimos mediante la clausula fq el filtro de geolocalización a aplicar al dato de coordenadas_p fijado con el parámetro sfield que tenga el valor deseado fijado con el parámetro pt que estén a menos de 2Km fijado con el parámetro d)
  http://www.zaragoza.es/buscador/select?wt=json&indent=true&q=*:*%20AND%20category:Restaurantes&omitHeader=true&fl=title,calle_t,telefono_s,coordenadas_p&fq=%7b!geofilt%7d&sfield=coordenadas_p&pt=41.658648,-0.909592&d=2
  El resultado se ha reducido a 130 restaurantes
- URL para JSON que muestra nombre, calle, tfno. y coordenadas de los restaurantes que están en un radio de 2Km de la estación ordenados por distancia (añadimos el parámetro sort para la función geodist() en orden ascendente)
  http://www.zaragoza.es/buscador/select?wt=json&indent=true&q=*:*%20AND%20category:Restaurantes&omitHeader=true&fl=title,calle_t,telefono_s,coordenadas_p&fq=%7b!geofilt%7d&sfield=coordenadas_p&pt=41.658648,-0.909592&d=2&sort=geodist()%20asc
  El resultado sigue siendo 130 restaurantes

- URL para JSON que muestra nombre, tfno. y coordenadas de los restaurantes que están en un radio de 2Km de la estación ordenados por distancia y que sean los 5 más próximos (añadimos el número de registros deseados a mostrar mediante los parámetros &start=0 para que empiece por el primero y &rows=5 para que sean solo 5)
  http://www.zaragoza.es/buscador/select?wt=json&indent=true&q=*:*%20AND%20category:Restaurantes&omitHeader=true&fl=title,calle_t,telefono_s,coordenadas_p&fq=%7b!geofilt%7d&sfield=coordenadas_p&pt=41.658648,-0.909592&d=2&sort=geodist()%20asc&start=0&rows=5
  El resultado sigue siendo 130 pero solo muestra los 5 primeros.
- URL para JSON que necesitas en tu aplicación para encontrar los 5 restaurantes más próximos a la estación (es la anterior eliminando la tabulación &indent)
  http://www.zaragoza.es/buscador/select?wt=json&q=*:*%20AND%20category:Restaurantes&omitHeader=true&fl=title,calle_t,telefono_s,coordenadas_p&fq=%7b!geofilt%7d&sfield=coordenadas_p&pt=41.658648,-0.909592&d=2&sort=geodist()%20asc&start=0&rows=5
- URL para XML que necesitas en tu aplicación para encontrar los 5 restaurantes más próximos a la estación (igual que anteriore pero sin wt=json)
  http://www.zaragoza.es/buscador/select?q=*:*%20AND%20category:Restaurantes&omitHeader=true&fl=title,calle_t,telefono_s,coordenadas_p&fq=%7b!geofilt%7d&sfield=coordenadas_p&pt=41.658648,-0.909592&d=2&sort=geodist()%20asc&start=0&rows=5

Alternativa 2:
Usando la API (primera pestaña), su ayuda y haciendo uso de Swagger UI

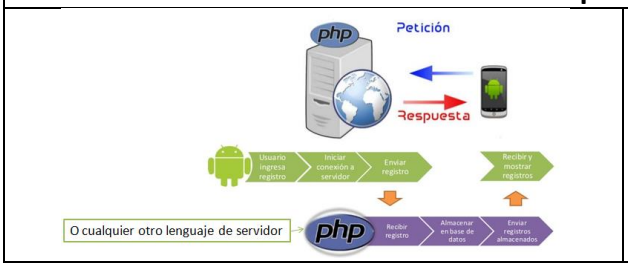


URL para obtener JSON:
http://www.zaragoza.es/sede/servicio/restaurante.json?fl=title,streetAddress,tel,geometry&srsname=wgs84&start=0&rows=5&distance=2000&point=-0.909592,41.658648

URL para obtener XML:
http://www.zaragoza.es/sede/servicio/restaurante.xml?fl=title,streetAddress,tel,geometry&srsname=wgs84&start=0&rows=5&distance=2000&point=-0.909592,41.658648

Android y Servicio Web REST

Android provee clientes HTTP para realizar conexiones y peticiones HTTP, pero no tiene (de momento) la característica de soportar REST como cliente, deberemos programar haciendo uso de las librerías para conexiones HTTP y determinando la acción a realizar según la URL accedida y el tipo de la petición (GET, POST, PUT o DELETE)

Desarrollo de la aplicación Android	
	

Repaso métodos de la AsyncTask :

- **doInBackground()**: es el que realiza la conexión y devuelve los resultados obtenidos de ella.
- **onPostExecute()**: toma los resultados anteriores y trabaja con ellos.
- **onPreExecute()**: no es obligatorio, pero es un buen método para mostrar un ProgressBar que avise al usuario del comienzo de la descarga.

Librerías para conexión

1. La clase **HttpClient** (del paquete org.apache.*)
   - Google recomienda no usarla en aplicaciones para dispositivos a partir de 2.3. porque no trabaja ya en su mejora:

     Prefer HttpURLConnection for new code

     Android includes two HTTP clients: `HttpURLConnection` and Apache HTTP Client. Both support HTTPS, streaming uploads and downloads, configurable timeouts, IPv6 and connection pooling. Apache HTTP client has fewer bugs in Android 2.2 (Froyo) and earlier releases. For Android 2.3 (Gingerbread) and later, `HttpURLConnection` is the best choice. Its simple API and small size makes it great fit for Android. Transparent compression and response caching reduce network use, improve speed and save battery. See the Android Developers Blog for a comparison of the two HTTP clients.

   - Se usa mucho en ejemplos encontrados en Internet porque simplifica el manejo de peticiones HTTP, pero AS la marca como obsoleta. Mejor no usarla por posibles agujeros de seguridad.
2. Desde Android 2.3, la clase HttpURLConnection del paquete java.net.* se usa para acceder a un recurso web.
3. La realización de las operaciones de red en Android puede ser "engorrosa" (hay que abrir y cerrar la conexión, programar el hilo/tarea asíncrona,…). Para simplificar estas operaciones existen bibliotecas de código abierto. Desde Google I/O 2013, Android Developers recomienda el uso de la librería **Volley**.

¿Entre HttpURLConnection o Volley qué utilizar?

Depende del tipo de formato en que estén los datos:

- si es JSON, lo menos engorroso es Volley
- si es XML, aunque tengamos que programar tareas asíncronas, lo mejor es HttpURLConnection, porque (hasta la fecha) no existe una manera ligera de trabajar con los datos y el tiempo que ahorramos por un lado lo perdemos usando librerías de terceros y programando clases.

Entonces, si puedes descargar los datos en los dos formatos, como en el Ayto. de Zaragoza, lo mejor es hacerlo en JSON; pero lamentablemente no siempre es posible, por ejemplo, AEMET ofrece sin clave los datos en XML o los RSS de las noticias de periódicos, etc.

Incluir en la lista blanca sitios inseguros

Desde Android 9, cualquier tráfico de red entre la aplicación y destinos inseguros debe incluirse explícitamente en una lista blanca. Ver Proteger a los usuarios con TLS por defecto en Android P.

Muchas de las url de obtención de datos son del tipo "http://..." y no del tipo "https://...".

Para poder usarlas deberemos incluirlas en la "lista blanca":

- En el directorio de recursos, hay que definir un documento XML para la configuración de seguridad de la red (por ejemplo res/xml/network_security_config.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain
includeSubdomains="true">www.zaragoza.es</domain>
    </domain-config>
</network-security-config>
```

- Y en el fichero AndroidManifest, en la etiqueta <application>, añadir el atributo :

```
<application
    android:networkSecurityConfig="@xml/network_security_config"
    ...
```

- Entonces se permitirá realizar solicitudes inseguras a cualquier dominio especificado dentro de ese archivo.

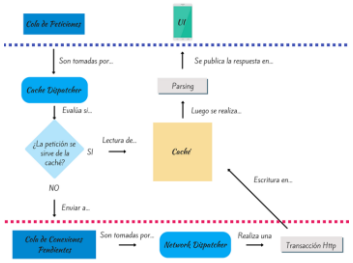
Usando la librería HTTP Volley

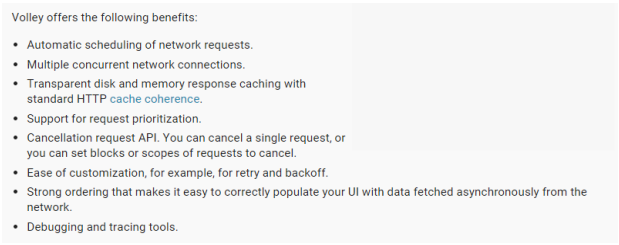
Android recomienda usar la librería HTTP Volley desde Google I/O 2013.

Volley es una librería desarrollada por Google para optimizar el envío de peticiones Http desde las aplicaciones Android hacia servidores externos.

Este componente actúa como una interfaz de alto nivel, liberando al programador de la administración de hilos y procesos tediosos de "parseo" de flujo de datos, para permitir publicar fácilmente resultados en el hilo principal.

Con Volley, solo necesitamos saber que tipo de respuesta esperamos (JSONObject, JSONArray, String etc.) y eso es todo, no necesitamos programar tareas asíncronas, ni conectar,…




Volley offers the following benefits:

- Automatic scheduling of network requests.
- Multiple concurrent network connections.
- Transparent disk and memory response caching with standard HTTP cache coherence.
- Support for request prioritization.
- Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
- Ease of customization, for example, for retry and backoff.
- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
- Debugging and tracing tools.

Pero… No es aconsejable usar Volley en descargas de datos pesados:

Volley is not suitable for large download or streaming operations, since Volley holds all responses in memory during parsing. For large download operations, consider using an alternative like `DownloadManager`.

Obtener volley.jar:

- Google recomienda es clonar el repositorio con la herramienta Git y luego construir un archivo .jar con base en este.
- Otro método más simple es usar el asistente de dependencias de AS:

```
implementation 'com.android.volley:volley:1.1.1'
```

Usando Volley

El uso de Volley comienza con creación de una cola de peticiones a través de un objeto de la clase RequestQueue que se encarga de gestionar automáticamente el envío de las peticiones, la administración de los hilos, la creación de la caché y la publicación de resultados en la UI. Creamos una instancia de la clase **RequestQueue** a partir del método **newRequestQueue** que recibe el contexto.

```
RequestQueue queue = Volley.newRequestQueue(getApplicationContext());
```

Para realizar peticiones en Volley podemos acudir a ciertos tipos que ya están estandarizados para uso frecuente:

- StringRequest: Este es el tipo más común, ya que permite solicitar un recurso con formato de texto plano, como son los documentos XML.
- ImageRequest: Como su nombre 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.

Recomendaciones "Patrón Singleton"

Normalmente una aplicación basada en servicios web necesita realizar peticiones a lo largo de todas sus actividades y componentes. Esta situación haría que hubiese que declarar colas de peticiones por todos lados o incluso pasar como parámetro la cola entre clases, lo cual llega a ser repetitivo, poco eficiente y confuso. Para desmontar este complejo enfoque, Google recomienda crear un Patrón Singleton que encapsule las funcionalidades necesarias de Volley. Este patrón se caracteriza por limitar el alcance de la clase a un solo objeto, restringiendo la instanciación de nuevos elementos. Básicamente esta clase debe contener como atributo la cola de peticiones y el contexto de la aplicación (ojo, no el contexto de la actividad, ya que es necesario establecer independencia de la interfaz, por si en algún momento existe un cambio de configuración, como la rotación de pantalla).

La clase Singleton es:

```
public final class MySingleton {
    // Atributos
    private static MySingleton singleton;
    private RequestQueue requestQueue;
    private static Context context;

    private MySingleton(Context context) {
        MySingleton.context = context;
        requestQueue = getRequestQueue();
    }

    public static synchronized MySingleton getInstance(Context context) {
        if (singleton == null) {
            singleton = new MySingleton(context);
        }
        return singleton;
    }

    public RequestQueue getRequestQueue() {
        if (requestQueue == null) {
            requestQueue = Volley.newRequestQueue(context.getApplicationContext());
        }
        return requestQueue;
    }

    public  void addToRequestQueue(Request req) {
        getRequestQueue().add(req);
    }

}
```

Obtención de imagen

Proyecto P_87_Volley_Imagen (igual que P_69_Asincrona_Descarga de Tareas costosas)

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });

        boolean acceso = hayRedDisponible();
        final TextView textView =  findViewById(R.id.textView);
        if (acceso) {
            final ProgressDialog pbar1 = ProgressDialog.show(this, "Por favor, espere", "Descargando datos...", true);
            String url = "http://i.imgur.com/7spzG.png";
            ImageRequest request = new ImageRequest(url,
                    new Response.Listener<Bitmap>() {
                        @Override
                        public void onResponse(Bitmap bitmap) {
                            ImageView mImageView = findViewById(R.id.imageView);
                            mImageView.setImageBitmap(bitmap);
                            textView.setText(R.string.descargado);
                        }
                    }, 0, 0, null, null,
                    new Response.ErrorListener() {
                        public void onErrorResponse(VolleyError error) {
                            textView.setText(R.string.imposible);
                        }
                    });
            RequestQueue queue = Volley.newRequestQueue(this);
            queue.add(request);
            pbar1.dismiss();
        }
        else
            textView.setText(R.string.imposible);
    }

    public boolean hayRedDisponible() {
        ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo redActiva = null;
        if (cm != null) {
            redActiva = cm.getActiveNetworkInfo();
        }
        return redActiva != null && redActiva.isAvailable() && redActiva.isConnected();
    }
```

Explicación:

Para construir el **ImageRequest** su constructor recibe:

- la URL a la que queremos acceder,
- la clase anónima **Response.Listener** que sobreescribe el método **onResponse()** si la respuesta fue existosa
- los valores 0, 0, null, null: indican que no modificamos el ancho, ni el alto, ni la escala de la imagen y que no mandamos ningún otro valor.
- la clase anonima **Response.ErrorListener()** que sobreescribe el método **onErrorResponse()** si la respuesta no tuvo éxito.

Solo queda ejecutar la solicitud, como hemos creado una instancia **RequestQueue** llamada "queue", usamos su método **add()** que recibe como parámetro el request preparado.

Si usamos el patrón Singlenton, entonces en la actividad que necesita añadir la solicitud a la cola de peticiones en vez de:

```
RequestQueue queue = Volley.newRequestQueue(this);
queue.add(request);
```

es más indicado:

```
// Access the RequestQueue through your singleton class.
MySingleton.getInstance(this).addToRequestQueue(request);
```

Obtención de datos en formato JSON

Proyecto P_88_SW_JSON: Listado de los 5 restaurantes más cercanos a la estación de RENFE de Zaragoza.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                                .setAction("Action", null).show();
            }
        });
        final RecyclerView recView = findViewById(R.id.recyclerView);
        recView.setHasFixedSize(true);
        RecyclerView.LayoutManager layoutmanager=new LinearLayoutManager(this,
        LinearLayoutManager.VERTICAL, false);
        recView.setLayoutManager(layoutmanager);
        boolean acceso = isNetworkAvailable();
        if (acceso) {
            String url =
"http://www.zaragoza.es/buscador/select?wt=json&q=*:*%20AND%20category:Restaurantes&fl=title,calle_t,
telefono_s,coordenadas_p&fq=%7b!geofilt%7d&sfield=coordenadas_p&pt=41.658648,-
0.909592&d=2&sort=geodist()%20asc&start=0&rows=5";
            JsonObjectRequest request = new JsonObjectRequest(Request.Method.GET,
                                url,
                                null,
                                new Response.Listener<JSONObject>() {
                                    @Override
                                    public void onResponse(JSONObject response) {
                                        parsearJSON(response, recView);
                                    }
                                },
                                new Response.ErrorListener() {
                                    @Override
                                    public void onErrorResponse(VolleyError error) {
                                        Toast.makeText(getApplicationContext(), R.string.imposible,
Toast.LENGTH_LONG).show();

                                    }
                                });
            MySingleton.getInstance(this).addToRequestQueue(request);
        }
    }
```

Explicación:

Para construir el **JsonObjectRequest** su constructor recibe:

- El método (GET, porque estamos solicitando obtener los datos)
- la URL a la que queremos acceder,
- la clase anónima **Response.Listener** que sobreescribe el método **onResponse()** si la respuesta fue existosa
- la clase anonima **Response.ErrorListener()** que sobreescribe el método **onErrorResponse()** si la respuesta no tuvo éxito.

Parseo de datos JSON

Visualizando la respuesta en el navegador, vemos que nos interesa el JSONObject response, dentro del que está el JSONArray docs dentro de cada uno de sus objetos, los elementos title y calle_t:

Como trabajamos con Volley, el formato descargado ya es JSONObject (podría ser JSONArray) y podemos usar las clases nativas JSONObject y JSONArray que nos proporcionan la colección de datos y los datos uno a uno respectivamente.

Algunos de estos datos podrían ser a su vez otros JSONArray y deberíamos repetir el proceso para analizar todos los datos. Como podemos ver este proceso es manual y puede llegar a ser tedioso pero nos permite todo el control que queramos sobre el proceso de parseo.

```
    private void parsearJSON(JSONObject devuelto, RecyclerView recyclerView) {
        // Necesitamos un ArrayList para los datos de los restaurantes
        ArrayList<Item> datos = new ArrayList<>();
        // Tenemos que analizar el JSON, en el navegador hemos visto que del conjunto de datos,
        // a nosotros nos interesan los que están en el objeto llamado response
        try {
            JSONObject objeto= devuelto.getJSONObject("response");
            // dentro de ese objeto nos interesa el array de nombre docs
            JSONArray datosQueNosInteresan = objeto.getJSONArray("docs");
            // bucle para cada uno de los restaurantes que aparecen
            for (int i = 0; i < datosQueNosInteresan.length(); i++) {
                // Dentro de ese conjunto hay un objeto JSON para cada restaurante
                JSONObject restaurante;
                try {
                    restaurante = datosQueNosInteresan.getJSONObject(i);
                    // De ese objeto por fin ya podemos extraer los datos
                    // que nos interesan a través de los nombres de sus etiquetas
                    String nombre = restaurante.getString("title");
                    String dire = restaurante.getString("calle_t");
                    // Añadimos la pareja anterior a la lista de restaurantes
                    datos.add(new Item(nombre,dire));
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
            //Construimos el adaptador y se lo añadimos al recycler
            MiAdaptador miAdaptador = new MiAdaptador(datos);
            recyclerView.setAdapter(miAdaptador);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

...
```

Obtención de datos en formato XML

Obtener los 5 restaurantes más próximos a la estación de RENFE de Zaragoza en formato XML

```
String url =
"http://www.zaragoza.es/buscador/select?wt=xml&q=*:*%20AND%20category:Restaurantes&omitHeader=true&fl
=title,calle_t,telefono_s,coordenadas_p&fq=%7b!geofilt%7d&sfield=coordenadas_p&pt=41.658648,-
0.909592&d=2&sort=geodist()%20asc&start=0&rows=5";
StringRequest stringRequest = new StringRequest(Request.Method.GET,
                        url,
                        new Response.Listener<String>() {
                            @Override
                            public void onResponse(String response) {
                                // aquí "parseo" de XML
                                textView.setText(R.string.posible);
                            }
                        },
                        new Response.ErrorListener() {
                            @Override
                            public void onErrorResponse(VolleyError error) {
                                textView.setText(R.string.imposible);
                            }
                        });
MySingleton.getInstance(this).addToRequestQueue(request);
```

Explicación:

Para construir el **StringRequest** su constructor recibe:

- El método (GET, porque estamos solicitando obtener los datos)
- la URL a la que queremos acceder,
- la clase anónima **Response.Listener** que sobreescribe el método **onResponse()** si la respuesta fue existosa
- la clase anonima **Response.ErrorListener()** que sobreescribe el método **onErrorResponse()** si la respuesta no tuvo éxito.

"Parseo de los datos": Aquí está la causa por la que te aconsejo usar HttpURLConnection para XML, porque como con Volley lo que obtenemos es un String, para poder analizarlo como XML, deberíamos

- convertir String a XML (serializar): Hay librerías de terceros http://simple.sourceforge.net/home.php

```
public void onResponse(String response) {
          // convert the String response to XML
          // if you use Simple, something like following should do it
            Serializer serializer = new Persister();
            serializer.read(ObjectType.class, response);
        }
```

- o bien, tratar la cadena, buscando etiquetas, etc

En cualquiera de los casos, muy laborioso.

Usando la clase HttpURLConnection

Es una librería del paquete java.net.*.Ya la hemos utilizado en alguno de nuestros proyectos (en el ejemplo de Servicios para descargar un fichero).

Iniciar conexión

El primer paso para iniciar la comunicación es abrir la conexión hacia el recurso alojado en el servidor. Para ello se usa el método **openConnection()** de la clase URL (se utiliza para definir un puntero a un recurso web).

El resultado que se obtenga debe ser casteado a HttpUrlConnection para que el cliente sea instanciado.

Ejemplo:

```
try {
    URL url = new URL("http://www.qqq.com/qaz");
    HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
    // acciones a realizar
} catch (Exception e) {
    e.printStackTrace();
}
```

Obtener datos con el método GET

Si se desea descargar datos desde la URL especificada simplemente se usa el método **getIntpuStream()** para obtener el flujo de datos asociado al recurso que se encuentra en esa dirección:

```
HttpURLConnection con = null;
try {
    URL url = new URL("http://10.0.2.2/paraAndroid/public/user/getAll");
    conexion = (HttpURLConnection)url.openConnection();
    conexion.setRequestMethod("GET"); // no es obligatorio, es la opción por defecto
    conexion.connect();  // no es obligatorio, es la opción por defecto
    InputStream lectura = conexion.getInputStream();
    // Acciones a realizar con el flujo de datos recibido
    conexion.disconnect();
} catch (Exception e) {
    …
```

Es importante que al finalizar la conexión se libere la memoria asociada a la instancia de la conexión realizada. Para ello se usa el método **disconnect()**, el cual pone a disposición de nuevo una futura reconexión

Acciones a realizar con el flujo de datos: "Parseo" del flujo de datos

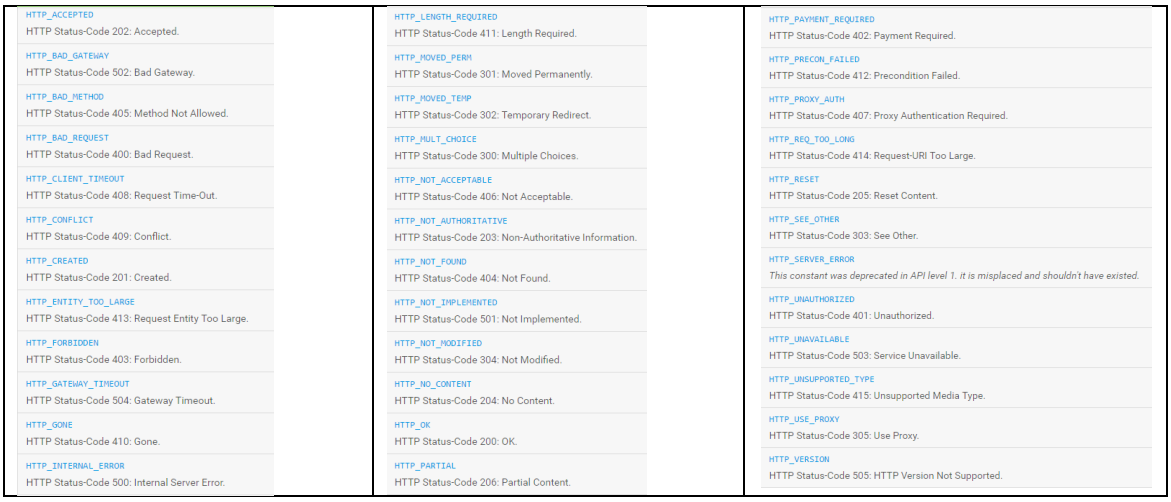
Un tipo InputStream debe ser decodificado para interpretar su contenido, ya sea texto plano, imagen, JSON, audio, etc. Dependiendo del objeto descargado se debe usar los métodos y técnicas correspondientes:

- **JSON/XML**
- Si deseas formatear un flujo a **Bitmap** puedes usar el método estático decodeStream() de la clase BitmapFactory, el cual recibe como parámetro un tipo InputStream (ya visto en ejercicio Tareas_costosas)
- Si en lugar de un bitmap, tienes **texto plano**, entonces puedes usar un objeto InputStreamReader, el cual puede convertir bytes a caracteres (char) y finalmente convertir a String

Otros métodos de la clase

El método **getResponseCode()** devuelve un entero que representa la respuesta a la petición del recurso:

```
int statusCode = conexion.getResponseCode();
if(statusCode!=200) {
        // acciones si no existe el recurso
}
else{
        // acciones si existe el recurso
}
```



HTTP_ACCEPTED
HTTP Status-Code 202: Accepted.

HTTP_BAD_GATEWAY
HTTP Status-Code 502: Bad Gateway.

HTTP_BAD_METHOD
HTTP Status-Code 405: Method Not Allowed.

HTTP_BAD_REQUEST
HTTP Status-Code 400: Bad Request.

HTTP_CLIENT_TIMEOUT
HTTP Status-Code 408: Request Time-Out.

HTTP_CONFLICT
HTTP Status-Code 409: Conflict.

HTTP_CREATED
HTTP Status-Code 201: Created.

HTTP_ENTITY_TOO_LARGE
HTTP Status-Code 413: Request Entity Too Large.

HTTP_FORBIDDEN
HTTP Status-Code 403: Forbidden.

HTTP_GATEWAY_TIMEOUT
HTTP Status-Code 504: Gateway Timeout.

HTTP_GONE
HTTP Status-Code 410: Gone.

HTTP_INTERNAL_ERROR
HTTP Status-Code 500: Internal Server Error.

HTTP_LENGTH_REQUIRED
HTTP Status-Code 411: Length Required.

HTTP_MOVED_PERM
HTTP Status-Code 301: Moved Permanently.

HTTP_MOVED_TEMP
HTTP Status-Code 302: Temporary Redirect.

HTTP_MULT_CHOICE
HTTP Status-Code 300: Multiple Choices.

HTTP_NOT_ACCEPTABLE
HTTP Status-Code 406: Not Acceptable.

HTTP_NOT_AUTHORITATIVE
HTTP Status-Code 203: Non-Authoritative Information.

HTTP_NOT_FOUND
HTTP Status-Code 404: Not Found.

HTTP_NOT_IMPLEMENTED
HTTP Status-Code 501: Not Implemented.

HTTP_NOT_MODIFIED
HTTP Status-Code 304: Not Modified.

HTTP_NO_CONTENT
HTTP Status-Code 204: No Content.

HTTP_OK
HTTP Status-Code 200: OK.

HTTP_PARTIAL
HTTP Status-Code 206: Partial Content.

HTTP_PAYMENT_REQUIRED
HTTP Status-Code 402: Payment Required.

HTTP_PRECON_FAILED
HTTP Status-Code 412: Precondition Failed.

HTTP_PROXY_AUTH
HTTP Status-Code 407: Proxy Authentication Required.

HTTP_REQ_TOO_LONG
HTTP Status-Code 414: Request-URI Too Large.

HTTP_RESET
HTTP Status-Code 205: Reset Content.

HTTP_SEE_OTHER
HTTP Status-Code 303: See Other.

HTTP_SERVER_ERROR
This constant was deprecated in API level 1. It is misplaced and shouldn't have existed.

HTTP_UNAUTHORIZED
HTTP Status-Code 401: Unauthorized.

HTTP_UNAVAILABLE
HTTP Status-Code 503: Service Unavailable.

HTTP_UNSUPPORTED_TYPE
HTTP Status-Code 415: Unsupported Media Type.

HTTP_USE_PROXY
HTTP Status-Code 305: Use Proxy.

HTTP_VERSION
HTTP Status-Code 505: HTTP Version Not Supported.

Una característica adicional de la clase HttpURLConnection es la capacidad de establecer los tiempos de caducidad de conexión y tiempos de caducidad de lectura a través de los métodos **setConnectTimeout()** y **setReadTimeout()**:

// Expirar a los 10 segundos si la conexión no se establece

```
conexion.setConnecTimeout(10000);
```

// Esperar solo 15 segundos para que finalice la lectura

```
conexion.setReadTimeout(15000);
```

Parseo de datos XML

Proyecto P_89_SW_XML: se desea visualizar en un listado los datos descargados en formato XML sobre ubicación y nº de bicis libres en las estaciones de "Bizi Zaragoza".

La URL de descarga es:

http://www.zaragoza.es/buscador/select?q=category:Bizi&rows=200

(Ojo!: esta URL es API antigua, puede que no se actualice, mejor usar API nueva)

Como ya hemos comentado utilizaremos la clase HttpUrlConnection y trabajaremos con tareas asíncronas.

Para poder analizar el fichero, es necesario conocer la manera en que guarda la información, para saber que datos son los que interesan leer.

Observemos que la estructura de la información suministrada es:


```
<doc>
  <int name="anclajesdisponibles_i">3</int>
  <int name="bicisdisponibles_i">18</int>
  <arr name="category">
    <str>Bizi</str>
  </arr>
  <str name="coordenadas_p">41.649912000000000000,-0.863471000000000000</str>
  <double name="coordenadas_p_0_coordinate">41.649912</double>
  <double name="coordenadas_p_1_coordinate">-0.863471</double>
  <str name="description">
    Datos de ocupación y localización de la estación bizi
  </str>
  <str name="estado_t">OPN</str>
  <str name="icon_t">
    http://www.zaragoza.es/contenidos/iconos/bizi/conbicis.png
  </str>
  <str name="id">bizi-101</str>
  <str name="language">es</str>
  <date name="last_modified">2018-02-03T19:06:00.338Z</date>
  <arr name="text">
    <str>C/ Doctor Iranzo - C/ Escultor Benlliure</str>
    <str>
      Datos de ocupación y localización de la estación bizi
    </str>
  </arr>
  <str name="texto_t">
    <ul><li>Estado: Operativa</li><li>Bicis disponibles: 18</li><li>Anclajes disponibles: 3</li></ul><p>Actualizado: 20:06</p>
  </str>
  <str name="tipocontenido_s">historico</str>
  <str name="title">C/ Doctor Iranzo - C/ Escultor Benlliure</str>
  <str name="uri">
    http://www.zaragoza.es/ciudad/viapublica/movilidad/bici/detalle_Bizi?oid=101
  </str>
  <double name="x_coordinate">678017.9803453928</double>
  <double name="y_coordinate">4613320.634481554</double>
</doc>
```


Cada estación es un elemento "doc" y de cada uno de ellos nos interesa la calle y el nº de bicis disponibles.

Lo primero que vamos a hacer es definir una clase para almacenar los datos que nos interesen (clase Item de tipo POJO). Nuestro objetivo final será devolver una lista de objetos de este tipo, con la información de todas las estaciones.

Por comodidad, vamos a almacenar todos los datos como cadenas de texto:

```
public class Item {
    private String libres;
    private String calle;

    public String getLibres() {
        return libres;
    }

    public void setLibres(String libres) {
        this.libres = libres;
    }

    public String getCalle() {
        return calle;
    }

    public void setCalle(String calle) {
        this.calle = calle;
    }
}
```

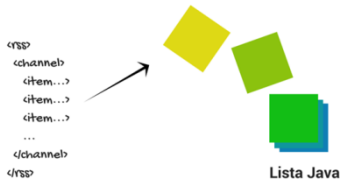
La clase MainActivity tampoco tiene ninguna novedad, trabaja con tarea asíncrona:

```
public class MainActivity extends AppCompatActivity {
    int size;
    String[] titulo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        …
        boolean acceso = isNetworkAvailable();
        if (acceso) {
            String urlDescarga = "http://www.zaragoza.es/buscador/select?q=category:Bizi&rows=130";
            new DescargaDatos().execute(urlDescarga);
        }
    }

    private class DescargaDatos extends AsyncTask<String, Void, ArrayList<Item>> {
        @Override
        protected ArrayList<Item> doInBackground(String... urlString) {
            ArrayList<Item> datos = null;
            try {
                URL url = new URL(urlString[0]);
                HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
                conexion.setReadTimeout(10000);
                conexion.setConnectTimeout(15000);
                conexion.setRequestMethod("GET");
                conexion.connect();
                InputStream stream = conexion.getInputStream();
                // PARSEO DEL CONTENIDO DESCARGADO
                datos = new ParserXML().parsear(stream);
                stream.close();
                conexion.disconnect();
            } catch (IOException ignored) {
            }
            return datos;
        }


        @Override
        protected void onPostExecute(ArrayList<Item> datos) {
            RecyclerView recyclerView=findViewById(R.id.recyclerView);
            recyclerView.setHasFixedSize(true);
            RecyclerView.LayoutManager layoutmanager=new
        LinearLayoutManager(getApplicationContext(),LinearLayoutManager.VERTICAL, false);
            recyclerView.setLayoutManager(layoutmanager);
            MiAdaptador miAdaptador = new MiAdaptador(datos);
            recyclerView.setAdapter(miAdaptador);
        }

    …
}
```

Descargamos

Es nuestra clase ParserXML, con su método parsear, la encargada de recorrer el fichero y extraer la información a la lista de objetos Item

Hay varias formas de analizar ("parsear") archivos XML, algunas genéricas de Java, otras propias de Android.



Android Developers recomienda **XmlPullParser** (interfaz que define la funcionalidad de análisis previsto en **XMLPULL V1 API**). XmlPullParser tiene métodos para obtener etiquetas, atributos, namespaces y contenidos CDATA.

La clase XmlPullParser basa su comportamiento de lectura en varios tipos de eventos. Estos nos servirán para usar condiciones que al ser verdaderas, obtengan los datos que requerimos. Los siguientes son los eventos más importantes:

Tipo de evento	Descripción
START_DOCUMENT	Este tipo de evento se presenta cuando se inicializa el parser.
START_TAG	Representa el inicio de una nueva etiqueta en el **DOM**. Esto permite que se lea el nombre de la etiqueta con getName() o los atributos de la etiqueta con getAttributeValue().
TEXT	Indica que se encontró el valor de la etiqueta y este puede ser leído en formato texto. Para ello usa el método getText().
END_TAG	Se da cuando el parser acaba de leer una etiqueta de cierre dentro de los datos XML. Si se desea obtener el nombre de dicha etiqueta, entonces se usa getName().
END_DOCUMENT	Indica que el parser acaba de encontrar el final del flujo XML.

(Algunos) Métodos de la clase:

- Crear nueva instancia del Parser: Se usa el método estático Xml.newPullParser() para crear una instancia de XmlPullParser.
  ```
  XmlPullParser parser = Xml.newPullParser();
  ```
- Indicar el flujo que alimenta al Parser: Se invoca el método setInput() que como primer parámetro recibe el flujo de datos tipo InputStream y como segundo parámetro se puede indicar el encoding del formato, como por ejemplo formato utf-8.
  ```
  parser.setInput(inputStream, null);
  ```
- Obtener el siguiente elemento o evento de parsing: Debido a que recorremos manualmente el archivo XML, se emplea el método next() para ir al próximo elemento o evento.
  ```
  parser.next()
  ```
- Obtener el tipo de evento actual en el que se encuentra el parser: con el método getEventType().
  ```
  parser.getEventType()
  ```

Existen también muchos métodos, sobre todo, los necesarios para trabajar con atributos.

Con estos conceptos claros, veamos nuestra clase:

```
class ParserXML {
    ArrayList<Item> parsear(InputStream inputStream) {
        ArrayList<Item> datos = null;
        XmlPullParser parser = Xml.newPullParser();
        try {
            String etiqueta;
            Item siguienteEstacion = null;
            parser.setInput(inputStream, null);



            int tipoEncontrado = parser.getEventType();
            while (tipoEncontrado != XmlPullParser.END_DOCUMENT) {
                switch (tipoEncontrado) {

                    case XmlPullParser.START_DOCUMENT:
                        datos = new ArrayList<>();
                        break;

                    case XmlPullParser.START_TAG:
                        etiqueta = parser.getName();
                        if (etiqueta.equals("doc")) {
                            siguienteEstacion = new Item();
                        }
                        if (etiqueta.equals("int") &&
(parser.getAttributeValue(0)).equals("bicisdisponibles_i")) {
                            if (siguienteEstacion != null) {
                                siguienteEstacion.setLibres(parser.nextText());
                            }
                        }
                        if (etiqueta.equals("str") && parser.getAttributeCount() > 0 &&
(parser.getAttributeValue(0)).equals("title")) {
                            if (siguienteEstacion != null) {
                                siguienteEstacion.setCalle(parser.nextText());
                            }
                        }
                        break;

                    case XmlPullParser.END_TAG:
                        etiqueta = parser.getName();
                        if (etiqueta.equals("doc") && siguienteEstacion != null) {
                            if (datos != null) {
                                datos.add(siguienteEstacion);
                            }
                        }
                        break;
                }
                tipoEncontrado = parser.next();
            }
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }
        return datos;
    }
```

Se crea el nuevo parser XmlPull y se establece el fichero de entrada en forma de stream [mediante XmlPull.newPullParser() y parser.setInput(...)].

Nos metemos en un bucle en el que iremos solicitando al parser en cada paso el siguiente evento encontrado en la lectura del XML. Para cada evento devuelto como resultado consultaremos su tipo mediante el método **parser.getEventType()** y responderemos con las acciones oportunas según dicho tipo.

Una vez identificado el tipo concreto de evento, podremos consultar el nombre de la etiqueta del elemento XML mediante **parser.getName()** y su texto correspondiente mediante **parser.nextText()**.

Usando servicios REST de terceros "grandes"

Para poder utilizar las API Rest en servicios web de terceros (twitter, ebay,…) en la mayoría de ellos nos obligan a tener cuenta de desarrolladores para conseguir posteriormente las claves de aplicación.

Todos ellos ofrecen documentación de la API:

- ebay
- facebook
- twitter

Ejemplos de proyectos (Son bastante viejos ☺!):

- Proyecto API ebay
- Proyecto API Twitter

Android y Servicio Web SOAP

Android no incluye "de serie" ningún tipo de soporte para el acceso a servicios web de tipo SOAP.

Es por esto por lo que hay que utilizar una librería externa para hacernos más fácil esta tarea. Entre la oferta actual, la opción más popular y más utilizada es la librería ksoap2-android.

Este framework permitirá de forma relativamente fácil y cómoda utilizar servicios web que utilicen el estándar SOAP.

Aunque están fuera de temario (por falta de tiempo, no de interés!), es interesante que leas los ejemplos de proyectos para servicios web SOAP:

Acceso desde Android a Web Service SOAP (hecho el servicio con Java)

Acceso desde Android a Web Service SOAP (hecho el servicio con .NET)

Prácticas propuestas

Realiza los ejercicios del fichero 22_Ejercicios_Datos_remotos