Conexiones Red: Json

JSON (JavaScript Object Notation) es un formato para intercambio de datos liviano, basado en texto e independiente del lenguaje de programación, que resulta fácil de escribir y leer tanto para los seres humanos como para las máquinas. JSON puede representar dos tipos estructurados: *objetos* y *matrices*. Un objeto es una colección no ordenada de cero o más pares de nombres/valores. Una matriz es una secuencia ordenada de cero o más valores. Los valores pueden ser cadenas, números, booleanos, nulos y estos dos tipos estructurados.

El Listado 1 constituye un ejemplo tomado de Wikipedia que muestra la representación JSON de un objeto que describe a una persona. El objeto tiene valores de cadena para nombre y apellido, un valor numérico para la edad, un valor de objeto que representa el domicilio de la persona y un valor de matriz de objetos de números telefónicos.

```
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address": {
        "streetAddress": "21 2nd  Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
     },
     "phoneNumbers": [
        {
            "type": "home",
            "number": "212  555-1234"
        },
        {
            "type": "fax",
            "number": "646  555-4567"
```

```
            }
        ]
}
```

Listado 1. Ejemplo de representación de un objeto en JSON

JSON suele utilizarse en aplicaciones Ajax, configuraciones, bases de datos y servicios web RESTful. Todos los sitios web populares ofrecen JSON como formato para intercambio de datos con sus servicios web RESTful.

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

Sr.No	Componentes y descripción
1	**Array([]** En un fichero JSON, se representa por corchetes
2	**Objetos({}** En ficheros JSON, se representan por llaves
3	**Key** A JSON object contains a key that is just a string. Pairs of key/value make up a JSON object Un Objeto JSON contiene una clave de tipo String. Un objeto JSON esta representado por Clave/Valor (Key, Value)
4	**Value** Contenido asociado a la clave (Key)

La API de modelos de objetos

La API de modelos de objetos es similar a la API de modelos de objetos de documento (DOM) para XML. Es una API de alto nivel que proporciona modelos de objetos inmutables para estructuras de objetos y matrices JSON. Estas estructuras JSON se representan como modelos de objetos usando los tipos de Java JsonObject y JsonArray. En la Tabla 1 se incluyen las clases e

interfaces principales de la API de modelos de objetos.
JsonObject suministra una vista Map para obtener acceso a la colección no ordenada de cero o más pares de nombres/valores del modelo. De modo similar, JsonArray ofrece una vista List para obtener acceso a la secuencia ordenada de cero o más valores del modelo.

Clase o interfaz	Descripción
Json	Contiene métodos estáticos para crear lectores, escritores, constructores de JSON y sus objetos de fábrica.
JsonGenerator	Escribe datos JSON en forma de stream, con un valor por vez.
JsonReader	Lee datos JSON de un stream y crea un modelo de objeto en la memoria.
JsonWriter	Escribe un modelo de objeto de la memoria en un stream.
JsonValue **JsonObject** **JsonArray** **JsonString** **JsonNumber**	Representan tipos de datos para valores en datos JSON.

Estos modelos de objetos también pueden crearse a partir de un origen de entrada (por ejemplo InputStream o Reader) usando la interfaz JsonReader. De modo similar, los modelos de objetos pueden escribirse en un origen de salida (por ejemplo OutputStream o Writer) usando la clase JsonWriter.
Por ejemplo, la API de Facebook devuelve los resultados de la búsqueda en el formato JSON que se muestra en el Listado 2. Más adelante veremos cómo hacer la petición.

```
{
   "data" : [
        { "from" : {  "name" : "xxx", ... }, "message" :  "yyy", ... },
        { "from" : {  "name" : "ppp", ... }, "message" :  "qqq", ... },
        ...
   ],
   ...
}
```

Listado 2. Representación JSON de una búsqueda en los posts públicos de Facebook

Podemos usar la API de modelos de objetos para obtener nombres y sus posts públicos relacionados con el término *java*. La línea 3 crea JsonObject para los resultados; la línea 5 itera respecto de cada resultado; y las líneas 6 a 9 obtienen el nombre de la persona que publicó el post y el post en sí, y los imprimen.

```
try {

        String string = “Data json” //Aquí va el string en format JSON

        JSONObject obj = new JsonObject(string);

        JSONArray results = obj.getJsonArray("data");

        for (int i = 0; i<results.length();i++)) {

            JSONObject obj = results.get(i);

             System.out.print(result.getJsonObject("from").getString("n
ame"));

             System.out.print(": ");

             System.out.println(result.getString("message", ""));

             System.out.println("-----------");

        }

   }catch()
```

Listado 3. Procesamiento de posts de Facebook con la API de modelos de objetos

Construcción de Json

Para la construcción de un objeto Json vamos añadiendo las partes que lo compone a través del método put(*clave*, *valor*) de igual manera con los Json Array, hasta formar el árbol genérico.

```
    JSONObject student1 = new JSONObject();
try {
    student1.put("id", "3");
    student1.put("name", "NAME OF STUDENT");
    student1.put("year", "3rd");
    student1.put("curriculum", "Arts");
    student1.put("birthday", "5/5/1993");

} catch (JSONException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

JSONObject student2 = new JSONObject();
try {
    student2.put("id", "2");
    student2.put("name", "NAME OF STUDENT2");
    student2.put("year", "4rd");
    student2.put("curriculum", "scicence");
    student2.put("birthday", "5/5/1993");

} catch (JSONException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

```
JSONArray jsonArray = new JSONArray();


jsonArray.put(student1);

jsonArray.put(student2);


JSONObject studentsObj = new JSONObject();

studentsObj.put("Students", jsonArray);


String jsonStr = studentsObj.toString();

System.out.println("jsonString: "+jsonStr);
```

Api Rest

REST se asienta sobre el protocolo HTTP como mecanismo de transporte entre cliente y servidor. Y en cuanto al formato de los datos transmitidos, no se impone ninguno en concreto, aunque lo más habitual, actualmente es intercambiar la información en formato XML o JSON. En este apartado vamos a ver el formato JSON.

Al trabajar con servicios web de tipo REST, las llamadas al servicio no se harán a través de una única URL, sino que se determinará la acción a realizar según la URL accedida y la acción HTTP utilizada para realizar la petición (GET, POST, PUT o DELETE).

Peticiones GET

Son las más sencillas como hemos visto en el apartado anterior, se utilizan para hacer consultas y recoger información. Para ello necesitamos una url, por ejemplo, para conseguir la información de un cliente concreto:

```
http://10.0.2.2:2731/Api/Clientes/Cliente/id_cliente
```

Dónde {{ id_cliente }} es la ID del cliente que queremos consultar su información. Pues bien, para conseguir esto construiremos un objeto URL a partir del String, y el HttpURLConnection realizará la petición, de la forma que sigue.

```
//Preparamos una variable para el string con el resultado JSON
String content = "";

// Creamos un objeto URL a partir del string pasado por parametro
URL urlFormed = new URL(url);

// Nos  creamos una conexión
HttpURLConnection myConnection =
        (HttpURLConnection) urlFormed.openConnection();

//Ponemos la cabecera con el tipo de formato (en este caso Json)
myConnection.setRequestProperty("Accept", "application/json");

// Leemos la respuesta: Si es correcto devuelve código 200 en otro caso es un error
if (myConnection.getResponseCode() == 200) {
```

```java
// Leemos la respuesta: Si es correcto devuelve código 200 en otro caso es un error
if (myConnection.getResponseCode() == 200) {

    //creamos un stringBuilder para una mejor eficiencia
    StringBuilder sb = new StringBuilder();

    //Conseguimos una referencia al flujo de entrada de datos de la conexión.
    InputStream responseBody = myConnection.getInputStream();

    //Creamos un InputStreamReader para pasar de bit a carácteres
    InputStreamReader responseBodyReader =
            new InputStreamReader(responseBody, "UTF-8");

    //Nos permite unir la secuencia de caracteres en una cadena
    BufferedReader bufferedReader = new BufferedReader(responseBodyReader);

    //Vamos sacando los string
    String line;
    while((line = bufferedReader.readLine()) != null){
        sb.append(line);
    }

    content = sb.toString();
    bufferedReader.close();


} else {
    // Aquí iria el manejo de errores
}
```

Peticiones POST

Las peticiones post se usan para la modificación de datos por lo que en la petición suele llevar información asociada. Usualmente si esta información es sensible debería encriptarse previamente. En este caso la construcción es más sencilla que las peticiones get ya que solo tenemos que formar el flujo de salida y para ello nos apoyamos en la clase PrintWriter que nos ayudar a formatear flujos de salidas de textos.

```
HttpURLConnection http = null;
int responseCode = -1;

try{
    URL url = new URL(strUrl);
    http = (HttpsURLConnection) url.openConnection();
    http.setRequestMethod("POST");
    http.setRequestProperty("Accept", "application/json");
    http.setDoOutput(true);


    //Nos permite formatear datos en un flujo de salida
    PrintWriter writer = new PrintWriter(http.getOutputStream());
    writer.print(data);
    writer.flush();


    responseCode = http.getResponseCode();



} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Biblioteca Volley

Como veis hacer peticiones puede llegar a ser bastante lioso, para ello existen bibliotecas que se encargan de manejar estas peticiones por nosotros. Es el caso de *Volley,* dedicada a realizar peticiones de una manera flexible sin necesidad de preocuparnos ni por la petición ni por la transformación de la respuesta.

Para la instalación debemos añadir la biblioteca al archivo app de gradle:

```
compile 'com.android.volley:volley:1.0.0'
```

Volley provee de las siguientes clases para hacer peticiones JSON:

JsonArrayRequest: Recoge la respuesta en un JSONArray desde resultado de una petición.

JsonObjectRequest: Recoge la respuesta en un JSONObject desde resultado de una petición.

StringRequest: Recoge el String que recibe como respuesta la petición.

Cola de peticiones.

Es importante indicar que las peticiones se gestionan a través de una cola.

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

Esta cola sigue un sistema Round-Robin, aunque existen métodos para poder crearnos nuestras propias colas y gestionar en qué orden gestionar las peticiones.

Una vez creada la petición se deberá añadir a la cola para que sea manejada por el sistema.

Un ejemplo de petición sería:

```
TextView mTxtDisplay;
ImageView mImageView;
mTxtDisplay = (TextView) findViewById(R.id.txtDisplay);
String url = "http://my-json-feed";
RequestQueue queue = Volley.newRequestQueue(this);


JsonObjectRequest jsObjRequest = new JsonObjectRequest
        (Request.Method.GET, url, null, new Response.Listener<JSONObject>() {

    @Override
    public void onResponse(JSONObject response) {
        mTxtDisplay.setText("Response: " + response.toString());
    }
}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
        // TODO Auto-generated method stub

    }
});

// Access the RequestQueue through your singleton class.
queue.add(jsObjRequest);
```

Como se puede observar, el constructor de JsonObjectRequest recibe cuatro parámetros:

- **Método de la petición**: GET, POST… Se usan los valores estáticos de la clase Request.Method.
- **URL**: String con la url a la que se le va a lanzar la petición.
- **JsonObject**: Datos asociados a la petición (Para peticiones POST, PUT..)
- **Respuesta correcta (listener)**: Listener que maneja la respuesta en caso de que la petición tenga éxito
- **Respuesta incorrecta(Listener)**: Listener que maneja la respuesta en caso de que la petición falle.