```
FileError: resource 'https://fonts.googleapis.com/css?family=Catamaran' g
  Error: getaddrinfo ENOTFOUND fonts.googleapis.com
 in input on line 13, column 1:
12
13 @import url(https://fonts.googleapis.com/css?family=Catamaran);
14
```

Apuntes

Descargar estos apuntes

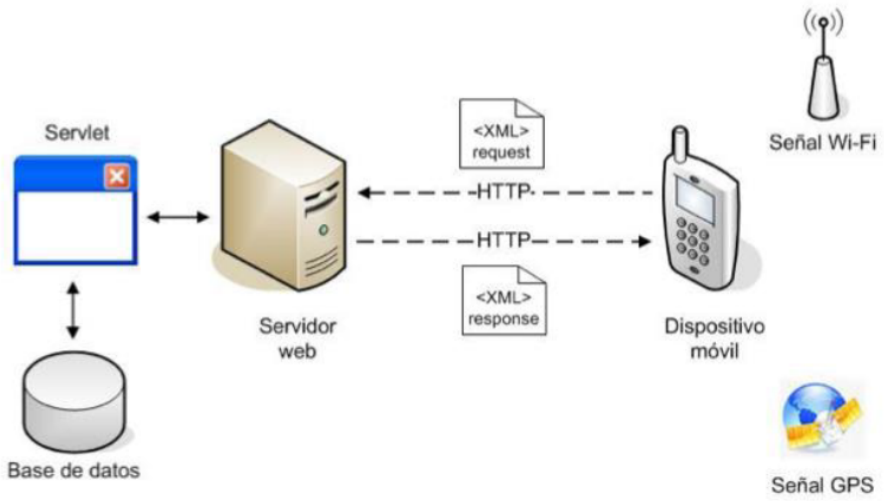
Tema 11. Persistencia de Datos

Índice

Acceso a bases de datos remotas

Introducción

Para obtener y guardar información de una base de datos remota, es necesario conectarse a un servidor donde se encontrará la BBDD. El esquema de funcionamiento lo podemos ver en la siguiente imagen:



En el servidor funcionan en realidad tres componentes básicos:

- Una base de datos, que almacena toda la información de los usuarios. Para la BBDD se puede utilizar MySQL, que es de licencia gratuita.

- Un servlet , que atiende la petición recibida, la procesa y envía la respuesta correspondiente. Un servlet no es más que un componente Java, generalmente pequeño e independiente de la plataforma.
- Un servidor web , donde reside y se ejecuta el servlet , y que permanece a la espera de conexiones HTTP entrantes.

Los formatos más utilizados para compartir información mediante estos servicios web son XML (y otros derivados) y JSON .

¿Qué es JSON?

JSON (Javascript Objec t Notation) es un formato ligero de intercambio de datos entre clientes y servidores, basado en la sintaxis de Javascript para representar estructuras en forma organizada. Es un formato en texto plano independiente de todo lenguaje de programación, es más, soporta el intercambio de datos en gran variedad de lenguajes de programación como PHP Python C++ C# Java y Ruby.
XML también puede usarse para el intercambio, pero debido a que su definición genera un DOM , el parseo se vuelve extenso y pesado. Además de ello XML debe usar Xpath para especificar rutas de elementos y atributos, por lo que demora la reconstrucción de la petición. En cambio JSON no requiere restricciones adicionales, simplemente se obtiene el texto plano y el engine de Javascript en los navegadores hace el trabajo de parsing sin ninguna complicación.

Tipos de datos en JSON

Similar a la estructuración de datos primitivos y complejos en los lenguajes de programación, JSON establece varios tipos de datos: cadenas, números, booleanos, arrays, objetos y valores null. El propósito es crear objetos que contengan varios atributos compuestos como pares clave valor. Donde la clave es un nombre que identifique el uso del valor que lo acompaña. Veamos un ejemplo:

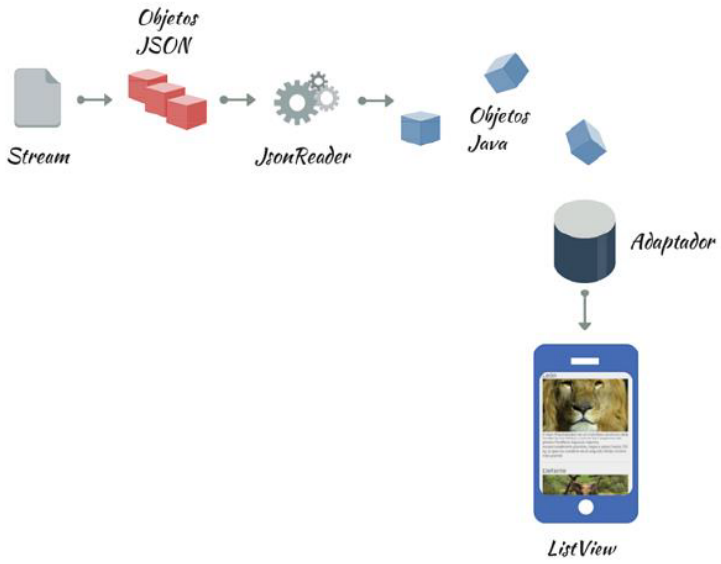
```
{
   "id": 101,
   "Nombre": "Carlos",
   "EstaActivo": true,
   "Notas": [2.3, 4.3, 5.0]
}
```

La anterior estructura es un objeto JSON compuesto por los datos de un estudiante. Los objetos JSON contienen sus atributos entre llaves `{}` , al igual que un bloque de código en Javascript, donde cada atributo debe ir separado por coma `,` para diferenciar cada par.
La sintaxis de los pares debe contener dos puntos `:` para dividir la clave del valor. El nombre del par debe tratarse como cadena y añadirle comillas dobles.
Si notas, este ejemplo trae un ejemplo de cada tipo de dato:

- El *Id* es de tipo entero, ya que contiene un número que representa el código del estudiante.
- El *Nombre* es un string. Usa comillas dobles para identificarlas.

- *EstaActivo* es un tipo booleano que representa si el estudiante se encuentra en la institución educativa o no. Usa las palabras reservadas true y false para declarar el valor.
- *Notas* es un arreglo de números reales. El conjunto de sus elementos debes incluirlos dentro de corchetes `[ ]` y separarlos por coma.

La idea es crear un mecanismo que permita recibir la información que contiene la base de datos externa en formato JSON hacia la aplicación. Con ello se parseará cada elemento y será interpretado en forma de objeto Java para integrar correctamente el aspecto en la interfaz de usuario.
La clase **JsonObject** de la librería **org.json.JSONObject**, puede interpretar datos con formato JSON y parsearlos a objetos Java o a la inversa.
Veamos una ilustración que muestra el proceso de parseo que será estudiado:



- Como puedes observar el origen de los datos es un servidor externo o hosting que hayas contratado como proveedor para tus servicios web. La aplicación web que realiza la gestión de encriptación de los datos a formato `JSON` puede ser `PHP`, `JavaScript`, `ASP.NET`, etc..
- Tu aplicación Android a través de un cliente realiza una petición a la dirección URL del recurso con el fin de obtener los datos. Ese flujo entrante debe interpretarse con ayuda de un parser personalizado que implementaran las clases que se utilizan para trabajar con `JSON`.
- El resultado final es un conjunto de datos adaptable al `API` de Android. Dependiendo de tus necesidades, puedes convertirlos en una lista de objetos estructurados que alimenten un adaptador que pueble un ListView o simplemente actualizar la base de datos local de tu aplicación en `SQLite`.

Acceso a bases de datos con Apirest y Retrofit

Retrofit es un cliente *REST* para Android y Java, desarrollado por Square, muy simple y fácil de aprender. Permite hacer peticiones `GET`, `POST`, `PUT`, `PATCH`, `DELETE` y `HEAD`; gestionar diferentes tipos de parámetros y parsear automáticamente la respuesta a un POJO (Plain Old Java Object).
Vamos a utilizar esta librería por su facilidad de manejo. Pero antes, aunque no es propio de esta asignatura, nos crearemos un sencillo ApiRest para poder acceder desde nuestra aplicación.

Definición del Servicio Rest

Consumo de un Servicio Rest desde Android

Existen diferentes librerías que nos permitirían consumir los servicios desde la App de Android, pero dada su facilidad vamos a utilizar las librerías: **Retrofit2** y **Gson**. Retrofit la utilizaremos para hacer peticiones y procesar las respuestas del APIRest, mientras que con Gson transformaremos los datos de JSON a los propios que utilice la aplicación.
Para ello añadiremos las siguientes líneas en el build.gradle de la app, y no olvides incluir permisos de internet: