

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Servicios en red

Codificación JSON

Lenguaje de programación backend

Conexiones HTTPS

Vamos a ver:

- Cómo funcionan las **conexiones HTTPS** y sus principales diferencias con respecto a las conexiones HTTP.
- El **esquema de tipo cliente/servidor** que, aunque ya puede que lo conozcas de otras asignaturas, en esta lo vamos a enfocar hacia las **aplicaciones Android**, ya que necesitan de una **conexión a un servidor externo**.
- Los **permisos** que necesitaremos otorgar a nuestras aplicaciones para que sean capaces de realizar conexiones HTTPS.
- Cómo podemos **obtener datos del tipo JSON** mediante una conexión HTTPS a un servidor externo, para poder **procesar en nuestras aplicaciones** Android.

Conexiones HTTP Y HTTPS

Cuando hablamos de conexiones HTTP, lo primero que posiblemente nos venga a la mente sea una visita a una página web a través de Internet, mediante un ordenador, pero, aparte de los ordenadores, hay numerosos dispositivos que también pueden realizar este tipo de conexiones, y como podremos adivinar, los smartphones son uno de ellos.

Hoy en día, el uso de un smartphone, sea del tipo que sea, es algo rutinario. Estos dispositivos tienen conexión a Internet, pudiendo utilizar aplicaciones muy comunes, como YouTube, WhatsApp, Telegram, Chrome e infinidad de juegos que nos podemos descargar y que hacen uso de similares tecnologías.

Cuando hablamos de **HTTP**, nos estamos refiriendo al protocolo que lleva su mismo nombre y que nos va a permitir realizar conexiones remotas entre dispositivos. Este protocolo quedó obsoleto hace mucho tiempo, aunque aún se siga utilizando, ya que **no es seguro**, es decir, **no cifra la información** antes de enviarla, lo que puede provocar que **alguien pueda acceder** a lo que se está enviando o recibiendo.

Esto se solucionó con la aparición del protocolo **HTTPS (Hypertext Transfer Protocol Secure)**, que sí utiliza un **cifrado SSL/TLS**, mucho más apropiado para el tráfico de información en este tipo de canales. Con este tipo de cifrado, se consigue que **ciertas informaciones**, como las contraseñas, **no puedan ser captadas por un tercero** de una forma tan sencilla como con HTTP.

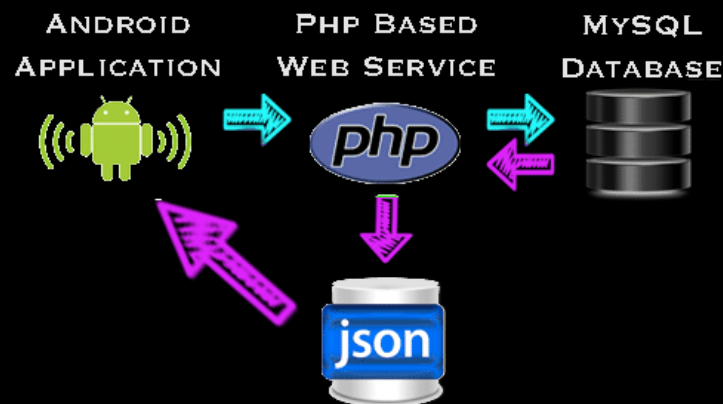
El **puerto** que utiliza el protocolo **HTTPS** es el **443** (HTTP usa el 80).

Como ya hemos comentado, absolutamente todos los smartphones actuales disponen de una **tarjeta de red integrada** que hará uso de este protocolo (entre otros) y que les permitirán realizar conexiones a Internet a unas velocidades muy altas.

Esquema cliente/servidor en aplicaciones Android

Todas las aplicaciones Android que vamos a realizar en esta unidad van a seguir el esquema que se muestra en la siguiente figura:

Esquema Android - PHP - JSON - MySQL



Este esquema consta de las siguientes **partes**:

1. **Aplicación Android**: Esta será nuestra aplicación instalada en nuestro smartphone.
2. **Servicio web basado en PHP**: Esta parte la componen todos los ficheros PHP que van a formar la **API REST**, y será con la que **accederemos a la parte de la base de datos**. Una API REST proporciona una forma estandarizada y eficiente para que **diferentes sistemas se comuniquen** a través de la web mediante el uso de los principios de REST y el protocolo HTTP.
3. **Base de datos MySQL**: Esta será la base de datos MySQL en la que estarán todas las tablas y datos en un **servidor remoto**.
4. **JSON**: Este será el **lenguaje intermedio** que permitirá la comunicación de la aplicación con la API REST en PHP.

El funcionamiento será el siguiente:

1. Cuando la **aplicación** necesite cierta información, **llamará a la API REST** en **PHP** mediante una **conexión HTTPS** a través de una **URL**.
2. La **API REST** realizará la operación solicitada por la aplicación **accediendo** a la **base de datos MySQL**.
3. Una vez que la API REST tenga la respuesta solicitada la **codificará en JSON** y la **devolverá** a nuestra aplicación Android.
4. La **aplicación Android** obtendrá dicha información codificada en **JSON**, la **decodificará** y podrá **tratar la información** obtenida.

Diferentes formatos, mismo esquema

*Ya comentamos en unidades pasadas que la información suele codificarse en JSON o en XML. En caso de que **necesitemos una codificación XML** podremos utilizar exactamente el **mismo esquema** cliente-servidor **que con JSON**, con la única variante de que la información **se devolverá en formato XML**.*

Transformando datos

Siempre que vayamos a usar **conexiones vía HTTPS** vamos a **comunicar una aplicación con el servidor utilizando JSON**, pero el formato JSON no es válido para trabajar en Android.

La actividad consiste por tanto en: dado un **fichero** con una serie de **datos de personas** codificado en **JSON**, **convertirlo a un ArrayList** y **mostrarlo en una lista optimizada**. Cada persona tiene un DNI, un nombre y unos apellidos.

En primer lugar, sí que debemos aprender a realizar este proceso, ya que lo vamos a necesitar realizar cada vez que nuestra aplicación realice una petición a un servidor, además, no es un proceso muy complicado, pero sí vital.

Este **proceso** se puede dividir en varias partes.

1. En primer lugar, deberemos crear una **clase** que represente la **información que vamos a obtener** del servidor, una clase Persona en este caso.
2. Una vez hecho, deberemos **recorrer el JSON**, que será un array, ya que podremos obtener varias personas, y **elemento a elemento** del array, en este caso, persona a persona, podremos ir **obteniendo todas sus propiedades** para **crear un objeto Persona** completo.
3. Por último, una vez tengamos la **persona creada**, deberemos **agregarla a un array**.

Puedes ver un ejemplo en pseudocódigo:

Transformando un array JSON en un array

```
// Convertimos el array json
desde i = 0 hasta json.tamaño
    dni = arrayjson.getElemento(i).obtenerDNI()
    nombre = arrayjson.getElemento(i).obtenerNombre()
    apellidos = arrayjson.getElemento(i).obtenerApellidos()
    persona = Persona(dni, nombre, apellidos)
    arrayPersona.agregar(persona)
// Mostramos el arrayjson.getElemento
desde i = 0 hasta arrayPersona.tamaño
    persona = arrayPersona.getElemento(i)
    mostrar(persona)
```

Permisos para conectar a Internet

El Sistema Operativo Android trabaja de una forma peculiar con respecto a sus aplicaciones, ya que estas se ejecutan de una forma distinta de las que se ejecutan, por ejemplo, en Windows. En este último sistema operativo, cualquier aplicación puede utilizar todos los recursos del ordenador con total libertad, cosa que no ocurre en los smartphones Android, por seguridad y debido a que los recursos son muy limitados.

En el desarrollo de aplicaciones Android, vamos a encontrarnos con que este sistema operativo usa un régimen de **permisos**, consistentes en **restricciones** que se adjudican a todas las aplicaciones, y que provocarán que estas no utilicen de forma indebida ciertos recursos como, por ejemplo, la cámara, el almacenamiento (tanto interno como externo), el bluetooth, el micrófono y un largo etcétera.

Cualquier aplicación Android necesitará que se le concedan permisos para poder utilizar los recursos de forma predeterminada. Esto se hará en el **momento de la instalación** de la misma, en el que nos aparecerá una lista de los permisos que dicha aplicación va a utilizar y que podremos examinar cuidadosamente antes de aceptar. Una vez **aceptados**, la aplicación **podrá utilizar** los recursos que indican sus permisos.

En nuestro caso, a la hora de desarrollar una aplicación, deberemos indicar los permisos en el manifiesto de la misma, fichero manifest.xml.

En el siguiente código xml podemos ver los permisos necesarios para que nuestra aplicación pueda tener una conexión a Internet.

Permisos de conexión a Internet

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ejemplovideo1">

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

...

```

Obtener JSON desde Android vía HTTP

Los conceptos teóricos estudiados hasta ahora ya nos permiten crear una aplicación Android que se **conecte a un servidor remoto**, donde exista una **API REST creada con PHP 7**, que accederá a una **base de datos MySQL** y codifica en **JSON**.

Para ello, vamos a **crear las siguientes clases**:

- **JSONParser**: Esta clase nos permitirá **obtener los datos con el formato JSON** de una **URL** específica, pudiendo pasarle parámetros por medio del **método POST**. Esta clase siempre se implementará de igual forma.
- **ServidorPHPException**: Debido a que el uso de **conexiones HTTPS** pueden provocar una gran lista de excepciones, vamos a crear una excepción que lanzaremos cuando alguna de estas ocurra, teniendo la gestión de errores centralizada en una sola excepción y no en muchas, que puede resultar muy tedioso.

Siempre que necesitemos utilizar **conexiones de tipo HTTPS**, necesitaremos usar el sistema de hilos, es decir, toda conexión del tipo HTTPS deberá ser **ejecutada mediante un hilo** independiente del resto de la aplicación.

Con la **clase JSONParser**, vamos a forzar a nuestra aplicación a **esperar** a que la **petición HTTPS** se realice para poder continuar. Esto **no es lo más adecuado**, pero para no sofisticar mucho las primeras aplicaciones que vamos a desarrollar, lo implementaremos así.

Dentro de la clase JSONParser, vamos a crear los siguientes **métodos**:

- **getJSONArrayFromUrl**: A este método le pasaremos la **URL** de la página que ejecuta el **servicio** en el servidor y un **array** con los **parámetros POST** necesarios. Nos devolverá un **objeto JSONArray** con todos los datos **devueltos** por el servicio en **formato JSON**.
- **getJSONObjectFromUrl**: De igual modo que el método anterior, a este le pasaremos la **URL** de la página que ejecuta el servicio en el **servidor** y un array con los **parámetros POST** necesarios. No obstante, en este caso, nos devolverá un **objeto JSONObject** con todos los datos **devueltos** por el servicio en **formato JSON**.
- **buildURL**: Este método privado **construye una URL** con la información que le facilitemos. Le pasaremos como parámetros la **URL base** y un **mapa** con todos los **parámetros** a pasar, **devolviéndonos la URL formateada** correctamente.

Cómo obtener datos JSON mediante una consulta HTTP mediante la biblioteca Volley:

Tendremos un servidor externo en una máquina virtual con XAMPP instalado:

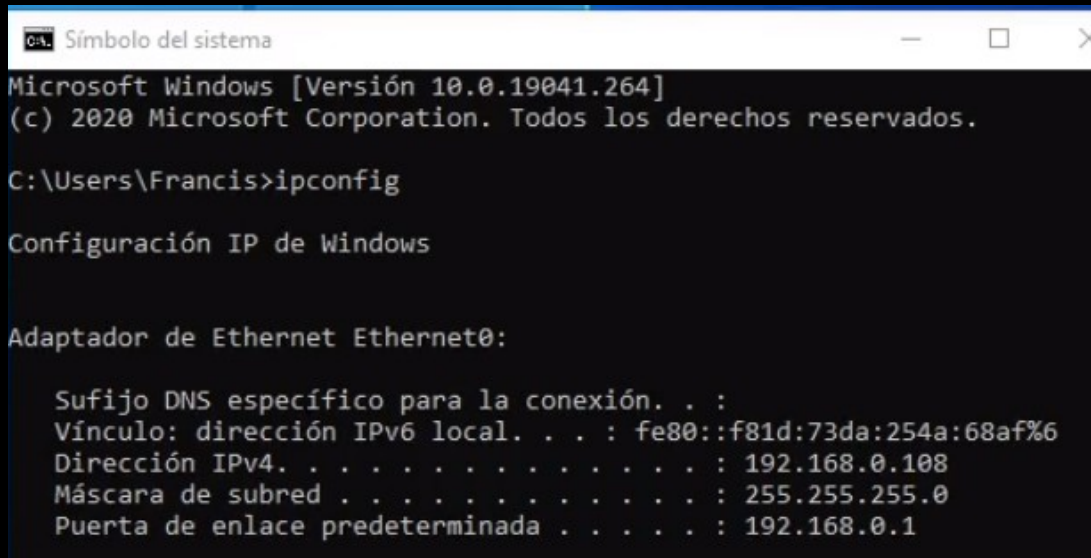


Con esta estructura de ficheros en la parte del servidor:

```
Símbolo del sistema
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Francis>tree C:\xampp\htdocs\personas /f
Listado de rutas de carpetas
El número de serie del volumen es 88D3-22A4
C:\XAMPP\HTDOCS\PERSONAS
    insertarPersona.php
    obtenerPersonaDNI.php
    obtenerTodasPersonas.php
    --controladores
        ControladorPersonas.php
    --datos
        ConexionBD.php
        login_mysql.php
        mensajes.php
    --modelos
        Persona.php
    --utilidades
        ExcepcionApi.php
    --vistas
        VistaApi.php
        VistaJson.php
```

Lo primero que debemos de configurar es la IP de nuestra máquina, ya que al ser un servidor debe de tener una IP estática. Mediante el comando ipconfig en Windows podremos ver nuestra IP, que en este caso es la Dirección IPV4 que vemos en la captura:



```
Microsoft Windows [Versión 10.0.19041.264]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Francis>ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet0:

    Sufijo DNS específico para la conexión. . . : 
    Vínculo: dirección IPv6 local. . . . . : fe80::f81d:73da:254a:68af%6
    Dirección IPv4. . . . . : 192.168.0.108
    Máscara de subred . . . . . : 255.255.255.0
    Puerta de enlace predeterminada . . . . . : 192.168.0.1
```

En la base de datos, que consultamos desde phpMyAdmin, vemos que tenemos la base de datos “personas” con una tabla llamada persona con los datos que se ven en la imagen:



Desde el fichero build.gradle (Module:app) colocaremos la biblioteca volley como se muestra a continuación:

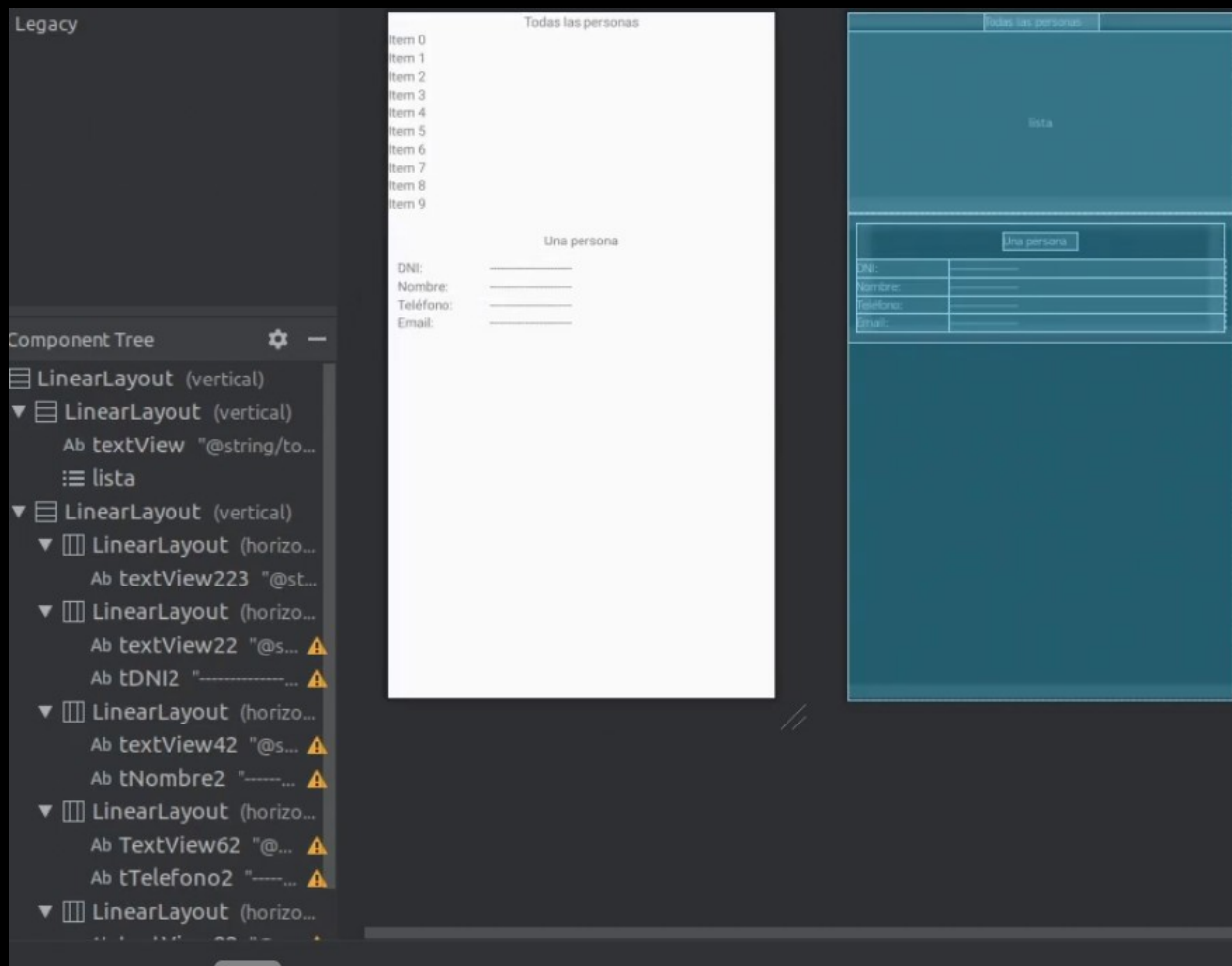
```
// Biblioteca Volley para la obtención de JSON desde una URL
implementation 'com.android.volley:volley:1.1.1'
```

Después tendremos que reconfigurar el proyecto para que se pueda usar esta biblioteca.
Lo siguiente que debemos hacer es conceder permisos de acceso a internet:

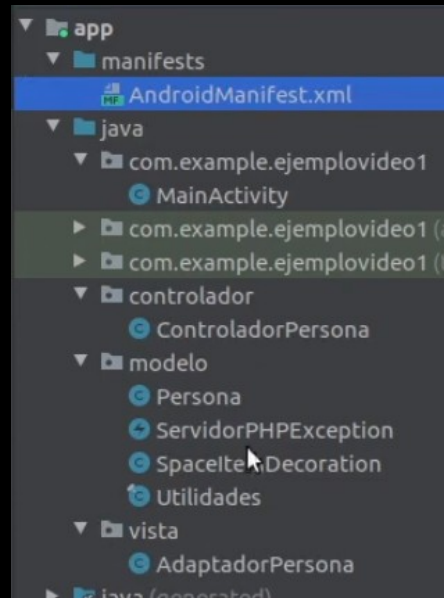
```
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Este fichero es el AndroidManifest.xml, dentro de la carpeta manifest.

La aplicación tiene una lista donde mostraremos a todas las personas y una serie de etiquetas con las que mostraremos los datos de cada persona, obtenida por su DNI:



Estamos usando el modelo vista controlador:



Dentro de “modelo” tenemos una clase llamada utilidades, en la que tenemos lo siguiente:

- Una serie de variables con los posibles resultados,
- Una variable con la URL de nuestro servidor, es decir, una url con la IP de nuestro servidor,
- Un método buildURL, que nos formateará una url en un formato válido.

```
public final class Utilidades
{
    // Variables con los posibles resultados
    public static final int RESULTADO_OK = 1;
    public static final int RESULTADO_ERROR = 2;
    public static final int RESULTADO_ERROR_DESCONOCIDO = 3;
    // Variable con la url de nuestro servidor, con la IP de nuestro servidor y la carpeta
    // personas con toda la información
    public static final String URLSERVIDOR = "http://192.168.0.107/personas/";

    /**
     * Crea una URL válida con parámetros
     * @param url URL base
     * @param params Parámetros para la URL
     * @return URL formateada con sus parámetros
     */
    public static String buildURL(String url, HashMap<String, String> params)
    {
```

```

Uri.Builder builder = Uri.parse(url).buildUpon();
if (params != null)
{
    for (Map.Entry<String, String> entry : params.entrySet())
    {
        builder.appendQueryParameter(entry.getKey(), entry.getValue());
    }
}
return builder.build().toString();
}
}

```

Además hemos creado también en “modelo” una clase Persona que será la que moldeará nuestro dato-objeto persona que tenemos en la base de datos:

```

public class Persona
{
    private String DNI, nombre, apellidos, telefono, email;

    public Persona(String DNI, String nombre, String apellidos, String telefono, String email) {
        this.DNI = DNI;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.telefono = telefono;
        this.email = email;
    }

    public String getDNI() {
        return DNI;
    }

    public void setDNI(String DNI) {
        this.DNI = DNI;
    }

    public String getNombre() {
        return nombre;
    }

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

```

```

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

public String getTelefono() {
    return telefono;
}

public void setTelefono(String telefono) {
    this.telefono = telefono;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}
}

```

Y tenemos también dentro del paquete “controlador” una clase ControladorPersona que será la encargada de gestionar las operaciones con personas, que tendrá lo siguiente:

- dos atributos de tipo String que guardarán la url del fichero .php que queramos llamar,
- Una variable de tipo Context, que será el contexto de la aplicación,
- Un RecyclerView,
- Un objeto de tipo AdaptadorPersona.

```

public class ControladorPersona {
    // atributos que guardarán la url del fichero que queramos llamar
    private final String URLOBTENERPERSONAS = URLSERVIDOR + "obtenerTodasPersonas.php";
    private final String URLOBTENERPERSONA = URLSERVIDOR + "obtenerPersonaDNI.php";

    private Context contexto;
    private RecyclerView lista;
    private AdaptadorPersona adaptador;
}

```

A nuestra clase le pasaremos en el constructor un objeto de tipo Context y la lista donde queremos mostrar las personas, ya que todo esto se hará mediante un hilo diferente al hilo principal de la aplicación, y deberemos de pasar los atributos gráficos donde queremos que se muestre la información, en este caso la lista y luego también las etiquetas:

```
public ControladorPersona(Context contexto, RecyclerView lista)
{
    this.contexto = contexto;
    this.lista = lista;
}
```

También hemos creado el método obtenerTodasPersonas, que utilizando la biblioteca Volley obtendrá mediante http los datos en nuestra url. Para ello deberemos crear una cola de peticiones de Volley e inicializarla con nuestro contexto y mediante un objeto de tipo JsonRequest podremos obtener la petición que queramos. Esto lo haremos a partir del try donde podemos obtener el resultado de nuestra url y saber si su estado es correcto. Si el estado es correcto obtendremos todos los datos en un objeto JSONArray e iremos obteniendo JSONObject por JSONObject y mediante los métodos get de su tipo de dato correspondiente obtendremos los datos de cada persona, agregándola a un array.

Una vez terminado esto, cerraremos la lista de una forma normal y mostramos el resultado mediante el método refrescar. Es importante que se haga en este punto ya que de lo contrario no podremos hacerlo al ser un hilo diferente:

```
/**
 * Obtiene todas las personas del servidor
 * @throws ServidorPHPException Esta excepción se lanzará si ocurre algún error en la conexión
 */
public void obtenerTodasPersonas() throws ServidorPHPException
{
    try
    {
        // Inicializo la cola de peticiones, utilizamos la biblioteca volley para obtener los datos en
        // nuestra url, para ello creamos la cola de volley
        RequestQueue colavolley = Volley.newRequestQueue(contexto);
        String url = URLOBTENERPERSONAS;
        // Mediante un objeto JsonRequestRequest podemos obtener la petición que queramos
        JsonRequestRequest jsonObjectRequest = new JsonRequestRequest(
            Request.Method.GET,
            url,
            null,
            new Response.Listener<JSONArray>()
            {
                @Override
                public void onResponse(JSONArray response)
```

```

{
    // Procesamos el JSONArray
    try
    {
        if( response != null )
        {
            int resultadoobtenido = response.getJSONObject(0).getInt("estado");
            //System.out.println("EL RESULTADO ES " + resultadoobtenido);

            switch(resultadoobtenido)
            {
                case RESULTADO_OK:
                    ArrayList<Persona> personas = new ArrayList<>();
                    JSONArray datospersonas =
                        response.getJSONObject(0).getJSONArray("mensaje");
                    for (int i = 0; i < datospersonas.length(); i++)
                    {
                        JSONObject per = datospersonas.getJSONObject(i);
                        String DNI = per.getString("DNI");
                        String nombre = per.getString("nombre");
                        String apellidos = per.getString("apellidos");
                        String telefono = per.getString("telefono");
                        String email = per.getString("email");
                        Persona persona =
                            new Persona(DNI, nombre, apellidos, telefono, email);
                        personas.add(persona);
                    }

                    // Hay que crear en la caperta values un fichero dims.xml y crear ahí
                    // list_space
                    lista.addItemDecoration(new SpacelItemDecoration(
                        contexto, R.dimen.list_space, true, true));
                    // Con esto el tamaño del recyclerview no cambiará
                    lista.setHasFixedSize(true);
                    // Creo un layoutManager para el recyclerview
                    LinearLayoutManager llm = new LinearLayoutManager(contexto);
                    lista.setLayoutManager(llm);

                    adaptador = new AdaptadorPersona(contexto, personas);
                    lista.setAdapter(adaptador);
                    // mostraremos aquí el resultado mediante el método refrescar
                    // Si no lo hacemos aquí no se podrá, al ser un hilo diferente
                    adaptador.refrescar();
                }
            }
        }
    }
}

```



```

        break;
    case RESULTADO_ERROR:
        throw new ServidorPHPException("Error, datos incorrectos.");
    case RESULTADO_ERROR_DESCONOCIDO:
        throw new ServidorPHPException("Error obteniendo los datos del
            servidor.");
    }
}
else
{
    throw new ServidorPHPException("Error obteniendo los datos del servidor.");
}
}
catch (JSONException | ServidorPHPException error)
{
    System.out.println("Error -> " + error.toString());
}
}
},
new Response.ErrorListener()
{
    @Override
    public void onErrorResponse(VolleyError error)
    {
        System.out.println("Error -> " + error.toString());
    }
}
);

```

Una vez terminemos agregamos la petición que hemos creado a la cola de la biblioteca Volley y se ejecutara en un hilo diferente:

```

// Agregamos la petición a la cola para que se ejecute, en un hilo diferente
colavolley.add(jsonObjectRequest);
}
catch (Exception error)
{
    throw new ServidorPHPException(error.toString());
}
}

/**
 * Obtiene una persona del servidor según su DNI
 * Al ejecutarse en un hilo aparte, debemos pasar DNI y todos los TextView donde se mostrarán los

```

```

* datos
* @param DNI DNI de la persona
* @param tDNI Etiqueta donde se mostrará el DNI
* @param tNombre Etiqueta donde se mostrará el nombre
* @param tTelefono Etiqueta donde se mostrará el teléfono
* @param tEmail Etiqueta donde se mostrara el email
* @throws ServidorPHPException Esta excepción se lanzará si ocurre algún error en la conexión
*/

```

El método obtenerPersonaDNI obtendrá los datos de una persona por su DNI y su funcionamiento es casi igual al método obtenerTodasPersonas.

Podemos ver que hemos pasado el DNI de la persona a buscar y todos los TextViews donde se mostrarán los datos, ya que al ejecutarse en un hilo diferente deberemos hacerlo así, igual que el el anterior método mostrábamos la lista.

En primer lugar volveremos a crear un cola de Volley para las peticiones y crearemos nuestro HashMap con los datos pertinentes, en este caso el DNI que será el DNI de la persona que queramos obtener. Construiremos el objeto url con el método buildURL de la clase Utilidades y una vez tengamos esto volvemos a crear un objeto del tipo JsonRequest de la misma forma.

```

public void obtenerPersonaDNI(String DNI, final TextView tDNI, final TextView tNombre, final
    TextView tTelefono, final TextView tEmail) throws ServidorPHPException
{
    try
    {
        // Inicializo la cola de peticiones
        RequestQueue colavolley = Volley.newRequestQueue(contexto);
        // Declaro el array HashMap de los parámetros con el DNI que qeremos obtener
        HashMap<String, String> parametros = new HashMap<>();
        // Meto los parámetros
        parametros.put("DNI", DNI);
        String urlfinal = Utilidades.buildURL(URLOBTENERPERSONA, parametros);
        JsonRequest jsonObjectRequest = new JsonRequest(
            Request.Method.GET,
            urlfinal,
            null,
            new Response.Listener<JSONArray>()
            {
                @Override
                public void onResponse(JSONArray response)
                {
                    try
                    {
                        if( response != null )
                        {

```

```

// Esta parte es muy importante: con estado y mensaje comprobamos si nuestras
// petición es correcta o no:
int resultadoobtenido = response.getJSONObject(0).getInt("estado");
//System.out.println("EL RESULTADO ES " + resultadoobtenido);
switch(resultadoobtenido)

```

Una vez se haya ejecutado, miramos si el estado es correcto, y de ser así obtenemos los datos de la persona y los mostramos en sus textos.

```

{
    case RESULTADO_OK:
        JSONArray datospersona =
            response.getJSONObject(0).getJSONArray("mensaje");
        JSONObject persona = datospersona.getJSONObject(0);
        tDNI.setText(persona.getString("DNI"));
        tNombre.setText(persona.getString("nombre") + " " +
            persona.getString("apellidos"));
        tTelefono.setText(persona.getString("telefono"));
        tEmail.setText(persona.getString("email"));
        break;
    case RESULTADO_ERROR:
        throw new ServidorPHPException("Error, datos incorrectos.");
    case RESULTADO_ERROR_DESCONOCIDO:
        throw new ServidorPHPException("Error obteniendo los datos del" +
            "servidor.");
}
}
else
{
    throw new ServidorPHPException("Error obteniendo los datos del servidor.");
}
}
catch (JSONException | ServidorPHPException e)
{
    System.out.println("Error -> " + e.toString());
}
}
},
new Response.ErrorListener()
{
    @Override
    public void onErrorResponse(VolleyError error)
    {
        System.out.println("Error -> " + error.toString());
    }
}

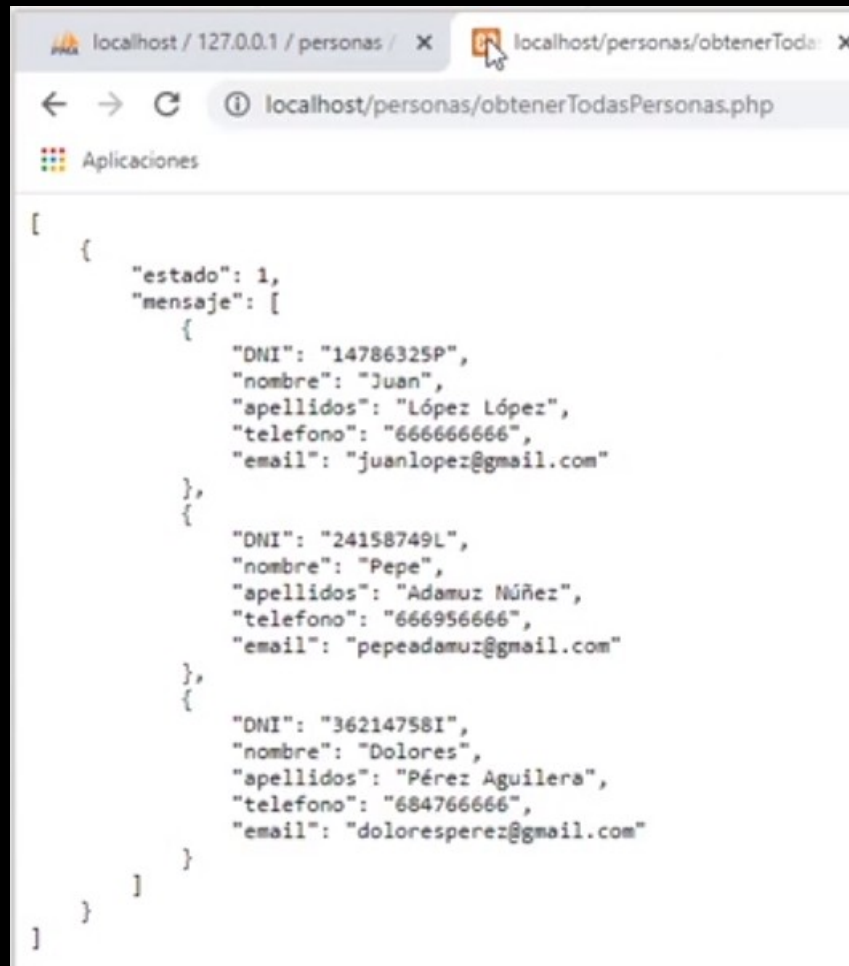
```

```
    }  
    );  
    // Agregamos la petición a la cola para que se ejecute
```

Cuando terminemos de realizar lo que queramos en el código, deberemos de agregar esa petición a la cola Volley. Esta parte es muy importante ya que mediante el estado y el mensaje sabremos si nuestra petición es correcta o no.

```
        colavolley.add(jsonObjectRequest);  
    }  
    catch(Exception e)  
    {  
        throw new ServidorPHPException("Error -> " + e.toString());  
    }  
}  
}
```

Si desde nuestro navegador, teniendo el servidor Apache corriendo con XAMPP, llamamos al fichero `obtenerTodasPersonas.php`, veremos en las respuestas siempre: en una parte el “estado” que nos dirá si se ejecutó bien, y en otra parte que es el “mensaje”, que si se ejecuta correctamente, estará el resultado de la petición, y si hay algún error, en el mensaje estará el error ocurrido.



Hecho todo esto solo nos queda crearnos en el MainActivity.java un controlador de Persona y llamar a sus dos métodos obtenerTodasPersonas y obtenerPersonaDNI:

```
public class MainActivity extends AppCompatActivity {

    private RecyclerView lista;
    private TextView tDNI2, tNombre2, tTelefono2, tEmail2;

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

        // **** Ligamos los recursos de la aplicación ****
        lista = findViewById(R.id.lista);
        tDNI2 = findViewById(R.id.tDNI2);
        tNombre2 = findViewById(R.id.tNombre2);
    }
}
```

```

tTelefono2 = findViewById(R.id.tTelefono2);
tEmail2 = findViewById(R.id.tEmail2);
// *****

ControladorPersona controladorp = new ControladorPersona(this, lista);

try
{
    controladorp.obtenerTodasPersonas();
    controladorp.obtenerPersonaDNI("24158749L", tDNI2, tNombre2, tTelefono2,
tEmail2);
}
catch (ServidorPHPException e)
{
    e.printStackTrace();
}

}
}

```

Al ejecutar la aplicación vemos los datos de todas las personas y los datos de una única persona con DNI 24158749L:

Ejemplo Volley	
Todas las personas	
DNI:	14786325P
Nombre:	Juan López López
Teléfono:	666666666
Email:	juanlopez@gmail.com
DNI:	24158749L
Nombre:	Pepe Adamuz Núñez
Teléfono:	666956666
Email:	pepeadamuz@gmail.com
DNI:	36214758I
Nombre:	Dolores Pérez Aguilera
Teléfono:	684766666
Email:	doloresperez@gmail.com
Una persona	
DNI:	24158749L
Nombre:	Pepe Adamuz Núñez
Teléfono:	666956666
Email:	pepeadamuz@gmail.com

Paso de parámetros en la petición

Ya hemos visto cómo podemos obtener los datos de un servidor remoto mediante una conexión HTTPS, pero ¿y si necesitamos **pasar ciertos parámetros a la petición HTTPS**?

Pongámonos en situación: necesitamos implementar un registro de usuarios, que se guardarán en la base de datos del servidor remoto. En este caso, nos vamos a ver obligados a **pedir la información de los datos al usuario** que se registra y, de alguna forma, deberemos **pasarla como parámetros** en la **petición HTTPS** que hagamos a nuestro servidor.

Esto es posible gracias a los **parámetros POST de PHP**. Estos ya los vimos en unidades pasadas y nos van a ofrecer la posibilidad de **pasar parámetros en las URL**, que podremos **rescatar en el código PHP** mediante la instrucción **`$_REQUEST`**.

Estos parámetros los podremos **pasar a los métodos `getJSONObjectFromURL` y `getJSONArrayFromURL`** como un **objeto de tipo `HashMap<String, String>`** en el que **guardaremos los datos** que queramos pasar a nuestra API REST.

Los **HashMap** nos van a permitir almacenar datos del tipo **clave-valor**, es decir, cada elemento del HashMap tendrá dos valores: una **clave que será única** y un **valor asociado** a dicha clave.

Esto lo podremos utilizar de forma que la **clave** sea el **nombre del parámetro** que queremos pasar y el **valor** sea el propio **valor del elemento**.

Podremos usar el **método `buildURL`**, al que le pasaremos la **dirección base**, es decir, sin parámetros, y un **objeto HashMap** con todos los **parámetros** que necesitemos. Este creará y formateará de forma automática la **dirección final**, devolviéndola lista para usar.

Importante: Hay que tener en cuenta que el **nombre del parámetro POST** debe tener exactamente el mismo nombre que el **valor que obtenemos con `$_REQUEST` en PHP**.

La importancia de la parte backend

Cuando se trata de desarrollar una **aplicación con parte backend**, la complejidad de las mismas se multiplica, ya que hay que realizar tanto la parte de servidor como la parte de la aplicación en sí y, además, cuidar que **ambas se comuniquen correctamente**.

Si realizamos de forma correcta la parte de backend (la del servidor) siguiendo las **pautas del Modelo-Vista-Controlador** y **mostrando** la información de forma correcta en **JSON**, con **códigos de error** que puedan indicar posibles errores en las conexiones, tendremos lo que se conoce como una **API REST totalmente funcional** que podremos aprovechar para que pueda ser **utilizada por otros dispositivos** como, por ejemplo, smartphones de Apple, aplicaciones de escritorio o incluso otros servicios web.

De hecho, la parte de **backend** suele desarrollarse por un **equipo diferente al que desarrolla las aplicaciones móviles** y es lo único que tienen que compartir con las aplicaciones. Es la forma de **comunicarse**, es decir, la parte de **mostrar los datos en formato JSON**.

Ejemplo completo de aplicación Android conectando a servidor remoto mediante conexiones HTTPS (con JSON y MYSQL)

El **servidor remoto** lo vamos a **simular** mediante un servidor local **XAMPP**, teniendo, además, las siguientes consideraciones:

- Al ser un servidor, deberemos **configurar la IP** de nuestra máquina como **fija**, para evitar estar cambiándola cada vez que reiniciemos.
- Tanto el ordenador que haga de **servidor** local y el **smartphone** con el que hagamos la prueba, ya sea un emulador o un dispositivo real, deberán estar **conectados a la misma red**. En **caso contrario**, **no serán visibles** entre ellos.

La funcionalidad que vamos a programar es la de **obtener todos los datos** de todas las personas que haya en el servidor, y **mostrarlas en una lista** dentro de nuestra aplicación.

También vamos a obtener los datos de **una única persona**, seleccionándola **mediante su DNI** y **mostrando** dichos datos en pantalla.

Para ello, vamos a utilizar el Modelo-Vista-Controlador tanto en la parte **backend** como en el **frontend**: nuestra aplicación Android.

La **base de datos** que vamos a utilizar será muy simple y solo tendrá **una tabla** donde se almacenarán los datos de las **personas** que obtendremos más adelante.

Continuando con la estructura del ejemplo articulado en el vídeo anterior, en el siguiente podrás ver el ejemplo completo de la aplicación Android conectándose al servidor. En el apartado de «Recursos del tema» podrás acceder a todo el código generado.

Este ejemplo es igual que el anterior, pero con un fichero en el paquete “modelo” llamado JSONParser.java:

Esta clase nos va a permitir, mediante los métodos `getJSONArrayFromUrl` y `getJSONObjectFromUrl` obtener datos en formato JSON de una url:

```
/**
 * Esta clase implementa la traducción JSON a un servidor web
 */
public class JSONParser
{
    public static final String TAG = "JSONParser";

    /**
     * Constructor de la clase
     */
    public JSONParser() {}
}
```

```

/**
 * Conecta con el servidor y devuelve un JSONArray con los datos obtenidos
 * @param direccionurl URL del servidor
 * @param parametros Parámetros de la consulta
 * @return JSONArray con los resultados de la consulta al servidor
 */
public JSONArray getJSONArrayFromUrl(String direccionurl, HashMap<String, String>
parametros) throws JSONException, IOException
{
    URL url;
    StringBuilder response = new StringBuilder();
    url = new URL(buildURL(direccionurl, parametros));

    HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
    urlConnection.setReadTimeout(15000);
    urlConnection.setConnectTimeout(15000);
    urlConnection.setRequestMethod("GET");
    urlConnection.setRequestProperty("Content-type", "application/json");

    int responseCode = urlConnection.getResponseCode();

    if(responseCode == HttpURLConnection.HTTP_OK)
    {
        InputStream in = new BufferedInputStream(urlConnection.getInputStream());
        BufferedReader reader = new BufferedReader(new InputStreamReader(in));

        String line;
        while ((line = reader.readLine()) != null)
        {
            response.append(line);
        }

    }
    urlConnection.disconnect();

    return new JSONArray(response.toString());
}

/**
 * Conecta con el servidor y devuelve un JSONObject con los datos obtenidos
 * @param direccionurl URL del servidor
 * @param parametros Parámetros de la consulta
 * @return JSONArray con los resultados de la consulta al servidor
 */

```

```

    public JSONObject getJSONObjectFromUrl(String direccionurl, HashMap<String, String>
parametros) throws JSONException, IOException
    {
        URL url;
        StringBuilder response = new StringBuilder();
        url = new URL(buildURL(direccionurl, parametros));

        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.setReadTimeout(15000);
        urlConnection.setConnectTimeout(15000);
        urlConnection.setRequestMethod("GET");
        urlConnection.setRequestProperty("Content-type", "application/json");

        int responseCode = urlConnection.getResponseCode();

        if(responseCode == HttpURLConnection.HTTP_OK)
        {
            InputStream in = new BufferedInputStream(urlConnection.getInputStream());
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));

            String line;
            while ((line = reader.readLine()) != null)
            {
                response.append(line);
            }

        }
        urlConnection.disconnect();

        return new JSONObject(response.toString());
    }

    /**
     * Crea una URL válida con parámetros
     * @param url URL base
     * @param params Parámetros para la URL
     * @return URL formateada con sus parámetros
     */
    private String buildURL(String url, HashMap<String, String> params)
    {
        Uri.Builder builder = Uri.parse(url).buildUpon();
        if (params != null)
        {
            for (Map.Entry<String, String> entry : params.entrySet())
            {

```

```
        builder.appendQueryParameter(entry.getKey(), entry.getValue());
    }
}
return builder.build().toString();
}
}
```

Comprobación de conexión a internet en app android

Podemos comprobar si estamos utilizando datos móviles en nuestra conexión a Internet, y en caso afirmativo preguntar al usuario si quiere seguir navegando aunque se esté usando la tarifa de datos.

Un ejemplo para comprobar si estamos usando conexión de datos para, después, preguntar al usuario lo tenemos en el siguiente ejemplo de código en, por ejemplo, nuestro MainActivity.java:

Ejemplo de comprobación de conexión con datos

```
ConnectivityManager cm = (ConnectivityManager)
getApplicationContext().getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

if (activeNetwork.getType() == ConnectivityManager.TYPE_MOBILE) {
    // Hay conexión de datos
    // Preguntar al usuario
}
```