


Open · Opened 3 days ago by  **Rubén Montero**

# Petición de red

## Resumen

- Veremos cómo usar la librería Volley para lanzar una solicitud de red simple
- Si el servidor responde correctamente, mostraremos un *Toast*
- Haremos `git add`, `git commit` y `git push` para subir los cambios al repositorio

## Descripción

Una aplicación móvil no es de mucha utilidad si no se conecta a Internet. Como mucho podremos programar una calculadora o un pequeño juego *offline*, pero, siendo realistas... ¿Cuántas *apps* instaladas en un dispositivo real acceden a Internet? ¡Todas!

## HTTP

[HyperText Transfer Protocol](#) es el [protocolo de aplicación](#) más empleado en la red de redes: *Internet*.

Tu navegador intercambia mensajes HTTP con algún servidor cada vez que visitas una página web.

Nosotros, desde nuestra aplicación, lanzaremos una solicitud HTTP al servidor <http://raspi> y esperaremos una respuesta.

Usaremos una librería llamada **Volley**. Hace falta tener la línea:

```
implementation 'com.android.volley:volley:1.2.1'
```

...en `build.gradle`. (Ya está añadido en nuestro proyecto)

## La tarea

Como primer paso, **abre** un terminal (cmd.exe) en tu ordenador y cambia el directorio activo hasta la ubicación de tu repositorio usando `cd`. Luego, **haz**:

```
git pull
```

...para traer los cambios de remoto a local, que contienen el *esqueleto* del proyecto de este *sprint*.

**Abre** el proyecto `android-clips/android-frontend-clips` desde Android Studio.

¡Ojo! En este *sprint* tendremos dos carpetas principales:

- `android-frontend-clips/` : Es la principal. Contiene el proyecto Android y trabajaremos sobre ella.
- `django-backend/` : Contiene código que es parte del proyecto, pero del *backend*. No trabajaremos aquí, pero puedes echar un vistazo a medida que avancemos

Después de abrir el proyecto, desde Android Studio, **abre** `java > com.afundacion.fp.clips > MainActivity.java` que ya nos resulta familiar:

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        // Aquí lanzaremos una petición HTTP  
    }  
}
```

## Lanzar una petición HTTP... ¿a dónde?

Al *endpoint* REST disponible en <http://raspi:8000/health>. Si abres dicha URL desde un navegador, verás una respuesta en formato JSON. La idea es solicitar ese JSON desde Android.

Como primer paso, **crea** una *instancia* de [JsonObjectRequest](#) <sup>1</sup>:

```
// Aquí Lanzaremos una petición HTTP
JsonObjectRequest request = new JsonObjectRequest();
```

Ahora, entre los paréntesis de `JsonObjectRequest()`, debemos pasar *cinco* parámetros que recibe el método constructor:

1. Una *constante* indicando el [método HTTP](#) de la petición. Nosotros siempre usaremos `Request.Method.GET`.
2. Un *String* indicando la URL.
3. El cuerpo de la petición. Es opcional. Siempre usaremos `null`.
4. Un *listener* en el que escribiremos el código responsable de manejar una *respuesta de éxito*.
5. Un *listener* en el que escribiremos el código responsable de manejar una *respuesta de error*.

Para empezar, **escribe** los tres primeros parámetros:

```
// Aquí Lanzaremos una petición HTTP
JsonObjectRequest request = new JsonObjectRequest(
    Request.Method.GET,
    Server.name + "/health",
    null,
    // ...
);
```

## Listeners

El siguiente *parámetro* del *constructor* es un `Response.Listener<JSONObject>`.

Si escribimos `new` y un *espacio en blanco*, Android Studio nos permitirá autocompletar y generar una *instancia anónima*. **Hazlo** así:

```
// Aquí Lanzaremos una petición HTTP
JsonObjectRequest request = new JsonObjectRequest(
    Request.Method.GET,
    Server.name + "/health",
    null,
    new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {

        }
    }
);
```

Aparecerá todo subrayado en *rojo*, porque el *constructor* recibe *cinco* parámetros, pero de momento sólo hemos indicado *cuatro*.

Para terminar, **añade** una *coma* ( , ) tras la última llave ( } ), y **escribe** `new`. Android Studio autocompletará generando un nuevo `Response.ErrorListener` *anónimo*:

```
// Aquí Lanzaremos una petición HTTP
JsonObjectRequest request = new JsonObjectRequest(
    Request.Method.GET,
    Server.name + "/health",
    null,
    new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {

        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    }
);
```

¡Perfecto! Hemos construido un `JsonObjectRequest`. Ahora, lo importante es *qué* código ponemos en `onResponse` y `onErrorResponse`.

En `onResponse`, **muestra** un *Toast* que indique `"GET OK"`:

```
// ...
new Response.Listener<JSONObject>() {
    @Override
    public void onResponse(JSONObject response) {
        // Aquí mostramos un Toast
        // que indique "GET OK"
    }
}, new Response.ErrorListener() {
// ...
```

## Cola de peticiones de red

**Añade** un atributo privado a `MainActivity`, de tipo `RequestQueue`:

```
public class MainActivity extends AppCompatActivity {
    private RequestQueue queue;

    // ...
```

...y, en la tercera línea de `onCreate`, **inicialízalo** empleando. Así:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.queue = Volley.newRequestQueue(context);
+
+
    // Aquí lanzaremos una petición HTTP
    // ...
```

Este *objeto* servirá para *lanzar* la petición `JsonObjectRequest` que hemos creado anteriormente. *Representa* una cola de peticiones de red. ¡Sólo hace falta invocar `queue.add`!

**Hazlo** al final de `onCreate`:

```
// ...
this.queue.add(request);
}
```

## Un último problemilla...

Si pruebas la aplicación, verás esta línea en los *logs*:

```
java.net.UnknownHostException: Unable to resolve host "raspi": No address associated with hostname
```

Nuestro emulador no es capaz de traducir `raspi` a una dirección IP válida de la misma manera que nuestro ordenador anfitrión sí lo hace. La IP de `raspi` es asignada por el DHCP de la red, que el emulador no *alcanza* a ver. Si `raspi` fuera un nombre de dominio públicamente accesible (como `www.google.es`), no habría esta complicación.

Ante esta limitación del emulador en lo relativo al DNS, debemos *sustituir* `raspi` en `Server.java` por una IP válida `X.X.X.X`.

**Sigue** las instrucciones en `Server.java` para hacer un *ping*, y luego, **sustituye** `raspi` por la IP correcta.

Una vez hecho esto, prueba de nuevo.

## ¡Funcional!

**¡Enhorabuena!** Has lanzado tu primera petición de red.

## Por último

Desde el terminal de Windows (cmd.exe), nos posicionamos en `android-clips/` y escribimos:

```
git add *
```



...para marcar los cambios del un nuevo *commit*. Después:

```
git commit -m "Tarea #20: Implementada petición simple a /health en android-clips"
```

...y por último subimos los cambios al repositorio remoto:

```
git push
```

No está de más visitar la página de GitLab (<https://raspi>) y verificar que el *commit* se ha subido.

1. `JsonObjectRequest` es una clase de la librería Volley que *representa* una petición HTTP. Hay *tres* posibilidades: `StringRequest` (Petición sencilla que espera un *texto plano* como respuesta), `JsonObjectRequest` (Espera un objeto JSON ( `{ ... }` ) como respuesta) y `JsonArrayRequest` (Espera un *array* JSON ( `[ ... ]` ) como respuesta). 
2. No, no falta sólo eso. Para que una aplicación pueda lanzar peticiones de red debe declarar el permiso `INTERNET` en el archivo *manifest*. Además, si el tráfico va por `http` y no `https`, será necesario `usesCleartextTraffic="true"`. Puedes verificar que ambas cuestiones ya está en el `AndroidManifest.xml` del proyecto. 



[Rubén Montero @ruben.montero](#) changed milestone to [%Sprint 2](#) 3 days ago