


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

Respuesta incorrecta del servidor, o no

Resumen

- Entenderemos cómo puede fallar una petición HTTP
- Mostraremos distintos *Toast* en distintos escenarios problemáticos

Descripción

En la tarea anterior hemos programado el *happy path* para mostrar un *Toast* con un mensaje del servidor. Pero es igual de importante programar los casos de error. Por ejemplo, ¿y si el usuario *no* tiene Internet?

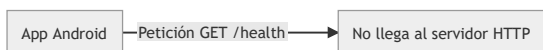
Con Volley, disponemos de `Response.ErrorListener`. Recuerda que lo habíamos añadido en la primera tarea:

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

    }
}
```

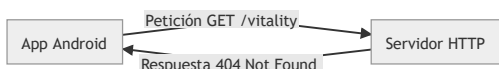
`onErrorResponse` es análogo a `onResponse`, pero para situaciones de error. Principalmente dos escenarios:

1) La conexión no se ha establecido



- El cliente no tiene conexión de red
- La red de destino no es accesible por cuestiones de *enrutamiento*
- El servidor HTTP recibe el mensaje, pero rechaza la conexión
- El servidor HTTP recibe el mensaje, pero lo ignora silenciosamente

2) La conexión funciona, pero el servidor responde un código de error



- El servidor HTTP acepta la conexión y responde con un código de error `4xx` ó `5xx`

Los *códigos de respuesta* HTTP se dividen en 5 familias:

1. `1xx`: Informativos
2. `2xx`: De éxito
3. `3xx`: De redirección
4. `4xx`: De error de cliente
5. `5xx`: De error de servidor

La tarea

En `MainActivity.java`, **añade** el siguiente `if-else` dentro del `onErrorResponse`:

```
@Override
public void onErrorResponse(VolleyError error) {
    if (error.networkResponse == null) {
        // Error: No se ha establecido la conexión

    } else {
        // Error: El servidor ha dado una respuesta de error

        // La siguiente variable contendrá el código HTTP,
        // por ejemplo 404, 500,...
```

```
        int serverCode = error.networkResponse.statusCode;
    }
}
```

(Como puedes observar, se estudia si `error.networkResponse` es nulo para distinguir entre los dos escenarios de error descritos anteriormente).

A continuación, dentro del `if`, **muestra** un `Toast` con el mensaje `"Server could not be reached"`

Para terminar, dentro del `else`, **muestra** un `Toast` con el mensaje `"Server KO: " + serverCode`. Como ves, se concatena el código de error del servidor al `String`, para que dicho código se muestre al usuario en el `Toast`.

¡Enhorabuena! Has controlado por primera vez los errores que pueden surgir de una petición HTTP. Lanza la *app* y verifica tu código.

¿Cómo verificarlo?

1. Para verificar que *no* hay conexión, desde el emulador o un dispositivo físico apaga la Wi-Fi en *Ajustes* y apaga el Uso de Datos Móviles en *Ajustes*. ¡Ya no hay Internet!
2. Para verificar un *código de error* puedes, *momentáneamente*, alterar el *endpoint*:

```
- Server.name + "/health",
+ Server.name + "/health2", // El servidor responderá 404 porque esta URL no existe
```

(Luego déjalo como estaba)

Por último

Sube tus cambios al repositorio en un nuevo *commit*.



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