


Open · Opened 2 days ago by  **Rubén Montero**

Autenticación

Resumen

- Persistiremos en las `SharedPreferences` el *Token de sesión* que recibimos al hacer un *Login* exitoso
- Lo mandaremos como *cabecera* HTTP en la petición `GET /users/<username>/status`

Descripción

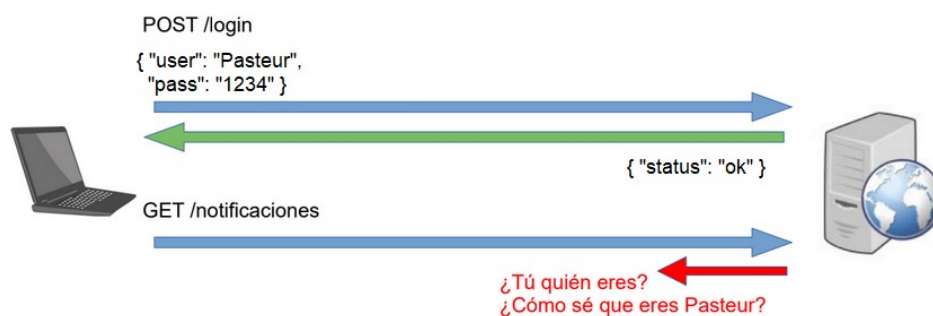
En la tarea anterior hemos implementado una petición `GET` a:

- `http://raspi:8000/users/<username>/status`

¡Pero la petición falla! Vemos **"Problema recibiendo el estado"** cuando iniciamos la pantalla principal. La verdad, es que el servidor REST nos está enviando una respuesta que indica esto:

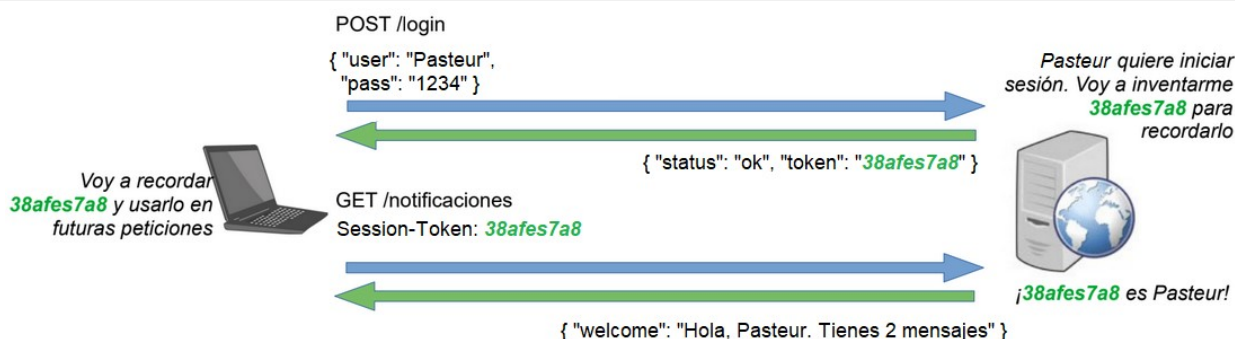
```
{
  "error": "Missing token header"
}
```

Las [APIs REST \(REpresentational State Transfer\)](#) no almacenan estado del cliente. *Toda* la información relevante viaja en la petición. Es decir, *no* podemos esperar que el servidor REST *nos recuerde*



Concepto de sesión: Autenticarse con un token

Cuando nos *logueamos* obtenemos un *Token de sesión*. Nuestra *app* debe *recordarlo* y *usarlo* en futuras peticiones.



En nuestro caso

¿Recuerdas *dónde* recibimos el *Token de sesión* del servidor?

¡Es en `LoginActivity.java`, dentro del `onResponse` que gestiona la respuesta a la petición `POST` de `Login`!

```
@Override
```

```
public void onResponse(JSONObject response) {
    String receivedToken;
    try {
        receivedToken = response.getString("sessionToken");
    } catch (JSONException e) {
        // Si el JSON de la respuesta NO contiene "sessionToken", vamos a lanzar
        // una RuntimeException para que la aplicación rompa.
        // En preferible que sea NOTORIO el problema del servidor, pues desde
        // la aplicación no podemos hacer nada. Estamos 'vendidos'.
        throw new RuntimeException(e);
    }
    // Si la respuesta está OK, mostramos un Toast
    // Esta línea asume que private Context context = this; está definido
    Toast.makeText(context, "Token: " + receivedToken, Toast.LENGTH_SHORT).show();

    /* ... */
}
```

En ese código, más abajo, ya tienes código que persiste el *nombre de usuario* (¡pero no el *Token*!)

```
SharedPreferences preferences = context.getSharedPreferences("SESSIONS_APP_PREFS", MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
editor.putString("VALID_USERNAME", username.getText().toString());
editor.commit();
```

La tarea

Para empezar, dentro del `onResponse` de `Login` en `LoginActivity.java`, **persiste** el *Token de sesión* (además del *nombre de usuario*). Así:

```
SharedPreferences preferences = context.getSharedPreferences("SESSIONS_APP_PREFS", MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
editor.putString("VALID_USERNAME", username.getText().toString());
+ editor.putString("VALID_TOKEN", receivedToken);
editor.commit();
```

Ahora, vamos a conseguir que ese *dato* viaje en la petición `GET /users/<username>/status` que tiene lugar en `StatusActivity.java`, mediante una cabecera HTTP.

Enviando una cabecera HTTP con Volley

Es un poco más rebuscado de lo que debería. No hay soporte *out-of-the-box* para hacerlo, así que debemos *crear una subclase* de `JsonObjectRequest` (ó `JSONArrayRequest`):

Primero, **añade** una nueva clase Java. **Dale** el nombre `JsonObjectRequestWithAuthentication.java`.

Luego, **declara** que tu clase hereda (`extends`) de `JsonObjectRequest`. Al hacerlo, saldrá un error porque no hay constructor *por defecto*. Este es un problema similar al que afrontábamos con nuestros `ViewHolder`. Para solventarlo, **haz click** en la bombilla roja de Android Studio, **selecciona** *Create constructor matching 'super'*, y **elige** la última opción (que corresponde al constructor más completo):

```
public class JsonObjectRequestWithAuthentication extends JsonObjectRequest {
    public JsonObjectRequestWithAuthentication(int method, String url, @Nullable JSONObject jsonRequest, Response.Listener<JSONObject> listener, @Nullable Response.ErrorListener errorListener) {
        super(method, url, jsonRequest, listener, errorListener);
    }
}
```

A continuación, **añade** un atributo privado tipo `Context` a la clase. También, **decláralo** en la firma del constructor que se acaba de autogenerar. Luego, **almacena** su valor en el atributo, como es habitual en un método constructor. Así:

```
public class JsonObjectRequestWithAuthentication extends JsonObjectRequest {
+   private Context context;

    public JsonObjectRequestWithAuthentication(int method,
        String url,
        @Nullable JSONObject jsonRequest,
        Response.Listener<JSONObject> listener,
        @Nullable Response.ErrorListener errorListener,
+       Context context) {
        super(method, url, jsonRequest, listener, errorListener);
+       this.context = context;
    }
}
```

```

    }
}

```

(Esto lo hacemos para recuperar el Token de sesión de las preferencias desde dentro de `JsonObjectRequestWithAuthentication`, en un método que implementaremos a continuación).

Sobrescribiendo `getHeaders`

Comienza escribiendo un nuevo método en la clase `JsonObjectRequestWithAuthentication`, tecleando `getHe`. **Dale** a ENTER para aplicar el autocompletado de Android Studio y tener:

```

public class JsonObjectRequestWithAuthentication extends JsonObjectRequest {
    private Context context;

    /* ... */

    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        return super.getHeaders();
    }
}

```

Este método que acabamos de sobrescribir (`@Override`), tiene esta documentación:

Returns a list of extra HTTP headers to go along with this request. (...)

(O sea que, ¡podemos usarlo para devolver (`return`) cabeceras HTTP!)

Ahora, **reemplaza** el `return super.getHeaders();` por las líneas necesarias para recuperar el Token de sesión de la persistencia. Así¹:

```

@Override
public Map<String, String> getHeaders() throws AuthFailureError {
    SharedPreferences preferences = context.getSharedPreferences("SESSIONS_APP_PREFS", Context.MODE_PRIVATE);
    String sessionToken = preferences.getString("VALID_TOKEN", null);

}

```

A continuación, **lanza** `AuthFailureError` si el Token de sesión no está persistido (es `null`). Si revisas la documentación completa de `getHeaders`, ¡veras que es lo adecuado!

```

if (sessionToken == null) {
    throw new AuthFailureError();
}

```

Y para terminar, **añade** estas líneas que hacen el `return` deseado:

```

HashMap<String, String> myHeaders = new HashMap<>();
myHeaders.put("Session-Token", sessionToken);
return myHeaders;

```

(Recordemos que, según [el estándar](#), las cabeceras HTTP viajan en formato clave-valor).

```

GET www.google.es
Accept-Language: es-ES
User-Agent: Mozilla/5.0

```

(Por ello, en `getHeaders` se devuelve un `Map<String, String>`, que es la versión Java de un diccionario clave-valor, ambos `String`. ¡Así conseguimos que nuestra cabecera se envíe!)

```

Session-Token: 234a5e6cb9

```

(`Session-Token` no es una cabecera estándar. La usamos así por diseño del API REST).

Con esto, método **completado**.

Lanzando la petición `GET status`, ahora con cabecera HTTP de sesión

En `StatusActivity.java`, **sustituye** la `JsonObjectRequest` por `JsonObjectRequestWithAuthentication`. Como es una subclase, puedes hacerlo.

Adicionalmente, **envía** un `context` al constructor:

```
private void retrieveStatus() {
    // Recuperamos el nombre de usuario de las preferencias
    SharedPreferences preferences = getSharedPreferences("SESSIONS_APP_PREFS", MODE_PRIVATE);
    String username = preferences.getString("VALID_USERNAME", null); // null será el valor por defecto
    // Mandaremos la petición GET
-   JsonObjectRequest request = new JsonObjectRequest(
+   JsonObjectRequestWithAuthentication request = new JsonObjectRequestWithAuthentication(
        Request.Method.GET,
        Server.name + "/users/" + username + "/status",
        null,
        new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                Toast.makeText(context, "Estado obtenido", Toast.LENGTH_LONG).show();
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Toast.makeText(context, "Problema recibiendo el estado", Toast.LENGTH_LONG).show();
            }
        }
    );
+   },
+   context
    );
    queue.add(request);
}
```

Si verificas la *app*, deberías ver el `Toast` que dice `"Estado obtenido"`.

Para completar la tarea, **añade** un atributo `TextView` a `StatusActivity.java`:

```
private TextView textViewStatus;
```

Inicialízalo con `findViewById` en `onCreate`. Debes referirte al `<TextView>` de `activity_status.xml` que pusimos con el texto `"Cargando"`. Si no tiene ID en el XML... ¡debes añadirlo!

Por último, en `onResponse` de `StatusActivity.java`, después del `Toast`, **muestra** en dicho `TextView` el mensaje en el JSON del servidor bajo la clave `"status"`:

```
textViewStatus.setText(response.getString("status"));
```

¡No olvides el necesario `try-catch` !

Cuando completes la tarea, **¡enhorabuena!** Has conseguido lanzar una petición con una cabecera HTTP *custom* en Volley para obtener un JSON así del servidor:

```
{
  "status": "Hey there! I'm using Sessions"
}
```

...y mostrarlo en la actividad principal de la *app*.

Por último

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

1. Sería lo suyo almacenar estos *Strings* como constantes `final` en alguna clase, en forma de variables estáticas. [🔗](#)



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

