Socket.io – Ejemplo mejorado

Un ejemplo más desarrollado de paso de mensajes con Socket.IO. Vamos a suponer que queremos solicitar del Servidor una lista con todas las Room en las que un Cliente está dado de alta. Para ello, tenemos que pasarle la ID del cliente, y la base de datos va a hacer una consulta que devolverá una Lista de Rooms.

La Petición al Servidor

Comenzamos por añadir el evento de escucha en el **SocketIOModule** del Servidor.

```
server.addEventListener(Events.ON_GET_ALL_CLIENT_REGISTERED_ROOMS.value, GetRoomsMessage.class,
        this.onGetAllCientRegisteredRooms());
```

Si te fijas, le he dicho que me llega un **GetRoomsMessage**. Esto no es más que un mensaje como en los ejemplos anteriores, pero le he querido dar otro nombre y he cambiado el atributo mensaje por clientID. Esto es porque necesito que me llegue la ID del cliente.

```
public class GetRoomsMessage extends AbstractMessage {

    private String clientID = null;

    public GetRoomsMessage(String clientID) {
        super();
```

Para que no me quede sucia la clase **SocketIOModule** con muchas líneas, yo lo tengo separado el código en otra clase con Singleton. **No tienes por qué hacerlo**. Es por claridad y porque el profe es un maniático.

```
private DataListener<GetRoomsMessage> onGetAllCientRegisteredRooms() {
    return ((client, data, ackSender) -> {
        SocketIOManager.getInstance().onGetAllCientRegisteredRooms(client, data);
    });
}
```

Vamos a ver cómo hacemos para leer el JSON de entrada y enviamos de vuelta el JSON con las Rooms. Yo me hago dos métodos porque así, en uno trato las excepciones y en el otro hago lo de los JSONs. **Tampoco tienes por qué hacerlo**. Manías del profe.

```
    public void onGetAllCientRegisteredRooms(SocketIOClient client, GetRoomsMessage data) {
        log.info("Client " + client.getRemoteAddress() + " request all rooms he is registere
        try {
            processGetAllCientRegisteredRooms(client, data);
        } catch (JsonSyntaxException e) {
            client.sendEvent(Events.ON_GET_ALL_CLIENT_REGISTERED_ROOMS_ERROR.value,
                    new PlainMessage(new Gson().toJson("Received JSON error")));
        } catch (Exception e) {
            client.sendEvent(Events.ON_GET_ALL_CLIENT_REGISTERED_ROOMS_ERROR.value,
                    new PlainMessage(new Gson().toJson("Server Error")));
        }
        log.info("Client " + client.getRemoteAddress() + " end request");
    }
```

O sea, mando llamo a processGetAllCientRegisteredRooms () y si algo va mal, pues ya mando excepciones. Pero esto no interesa. Vamos al grano: objeto a JSON y JSON a objeto, tanto en Java como en Kotlin.

```
private void processGetAllCientRegisteredRooms(SocketIOClient client, GetRoomsMessage data) {
    // {"clientID":"1"}
    JsonObject JsonObject = new Gson().fromJson(data.getClientID(), JsonObject.class);
    String clientID = JsonObject.get(JSONMapping.CLIENT_ID.value).getAsString();

    OperationResult<List<Room>> operationResult = new RoomDBManagerFake()
            .getAllRoomsFromClient(Integer.parseInt(clientID));

    Gson gson = new Gson();
    if (operationResult.isSuccess()) {
        RoomsMessage roomsMessage = new RoomsMessage(operationResult.getData());
        client.sendEvent(Events.ON_GET_ALL_CLIENT_REGISTERED_ROOMS_RESPONSE.value, gson.toJson(roomsMessage));
    } else {
        client.sendEvent(Events.ON_GET_ALL_CLIENT_REGISTERED_ROOMS_ERROR.value,
                gson.toJson(new ErrorMessage(operationResult.getErrorMessage())));
    }
}
```

Mira la primera parte del método **processGetAllCientRegisteredRooms ():**

```
// {"clientID":"1"}
JsonObject JsonObject = new Gson().fromJson(data.getClientID(), JsonObject.class);
String clientID = JsonObject.get(JSONMapping.CLIENT_ID.value).getAsString();
```

Lo que espero que me llegue es este JSON tan sencillo. No me complico la vida, le saco el valor de la ID a pelo. ¿Cómo he mandado esto desde la app móvil?

```
fun sendMessageGetAllRegisteredRooms(message: GetRoomsMessage) {
    val messageJSON = Gson().toJson(message)
    socket.emit(SocketEvents.ON_GET_ALL_CLIENT_REGISTERED_ROOMS.value, messageJSON)
    Log.d (tag, message.toString())
}
```

¿Y qué es **GetRoomsMessage**? La misma clase que en el Servidor, pero en Kotlin:

```kotlin
data class GetRoomsMessage (private val clientID: String)
```

```java
public class GetRoomsMessage {

    private String clientID = null;

    public GetRoomsMessage(String clientID) {
        super();
```

Es decir:

1) Tengo claro qué necesito pedirle al server (la ID del cliente, pero puede ser cualquier cosa).
2) Hago un objeto con eso (el GetRoomsMessage**)**
3) Lo convierto en JSON a pelo con Gson. (Lo serializo)
4) Lo emito al Server
5) El Server lo procesa (Lo deserializo)

La Respuesta del Servidor

Nos quedamos en el server. La segunda parte del **processGetAllCientRegisteredRooms ()** es un acceso a Hibernate. Ignóralo, porque es código de ejemplo y siempre devuelve una Lista <Room> cargado. Si la cosa ha ido bien, el programa se va por el isSuccess (). Esto es lo que nos interesa:

```java
if (operationResult.isSuccess()) {
    RoomsMessage roomsMessage = new RoomsMessage(operationResult.getData());
    client.sendEvent(Events.ON_GET_ALL_CLIENT_REGISTERED_ROOMS_RESPONSE.value, gson.toJson(roomsMessage));
} else {
```

¿Qué hacemos? En **operationResult** tenemos la lista de las Rooms (Lista <Room>). Se la metemos en un **RoomsMessage**, que es otro mensaje como los anteriores que tiene dentro:

```java
public class RoomsMessage {

    private List<Room> rooms = null;

    public RoomsMessage(List<Room> rooms) {
        super();
```

Convertimos el **RoomsMessage** en JSON y lo emitimos como respuesta.

Ahora, volvemos a la App móvil. ¿Qué nos falta? En efecto: lo mismo que en el Server. La clase **Room** (que es una Entity de BBDD) y la clase **RoomsMessage**. Las pasamos a Kotlin.

```
data class RoomsMessage (val rooms: List<Room>)
```

```
data class Room (val id: Int,
                 val name : String,
                 val type : String,
                 val description : String ){
}
```

Así que ahora el Gson puede hacer el paso JSON – Objeto fácilmente:

```
socket.on(SocketEvents.ON_GET_ALL_CLIENT_REGISTERED_ROOMS_RESPONSE.value) { args ->
    val response = args[0] as String

    val type = object : TypeToken<RoomsMessage>() {}.type
    val roomsMessage: RoomsMessage = Gson().fromJson(response, type)

    activity.findViewById<TextView>(R.id.textView).append("\n" + roomsMessage)
    Log.d (tag,  msg: "The answer: $roomsMessage")
}
```

Le decimos: saca args [0] como String porque es un JSON. Ese JSON es en realidad de tipo **RoomsMessage**, así que me lo conviertes. En el ejemplo luego simplemente hace un toString y lo vuelca en el TextView, pero ya tienes el listado de las Rooms listo para usar como necesites.

Es decir:

1) Tengo claro qué voy a devolver (la lista de Rooms).
2) Hago un objeto con eso (el RoomsMessage**)**
3) Lo convierto en JSON a pelo con Gson (lo serializo)
4) Lo emito al Cliente
5) El Cliente lo procesa porque tiene sus Room y RoomsMessage para deserializarlo.

Como curiosidad, el JSON que le llega de respuesta a la App es:

RoomsMessage(rooms=[Room(id=1, name=DAM, type=Public, description=A public room for DAM), Room(id=2, name=DAW, type=Public, description=A public room for DAW)])