Seminario 2: Websockets

Blanco Villorejo, Saúl

Matagne, Miro-Manuel

Muñoz Crespo, Guillermo

https://gitlab.com/m3642/sad-act-websocket

**Descripción**

En esta tarea hemos modificado el código de un servicio de chat que usaba websocket para implementar nuevas características.

**Distribuir un mensaje a todos los usuarios conectados cuando alguien se   
conecta o se desconecta**

Para ello, el servidor debe emitir un mensaje a través de cada uno de los sockets activos cuando un usuario se conecta.

io.on(“connection”, (socket) => {

…

io.emit(“chat message”, “Nuevo usuario conectado”);

...

)

**Añadir soporte a nicknames**

Para añadir soporte a los nicknames, se ha creado en el servidor una lista de usuarios conectados a través de un objeto de javascript. Cada uno de los miembros del objeto es un par clave-valor donde la clave es el ID del socket y el valor es el nick del usuario.ç

(Rellenar).

**No enviar el mensaje al usuario que ha enviado su mensaje. El usuario deberá visualizarlo localmente nada más se pulsa Intro.**

Websockets nos permite enviar un mensaje a todos los usuarios activos menos al usuario que ha enviado un mensaje a través de socket.broadcast.emit().

socket.on(“chat message”, (msg) =>

{

socket.broadcast.emit(

“chat message”,

`${connectedUsers[socket.id]}: ${msg}`

);

});

Además, el código del navegador deberá añadir un nuevo mensaje a la interfaz sin esperar a recibir respuesta del servidor.

$(‘#messages’).append($(‘<li>’).text(‘Yo: ’ + input));

**Añadir funcionalidad “{usuario} está escribiendo”**

Para añadir esta funcionalidad se ha creado un nuevo tipo de mensaje “typing”.

La funcionalidad del cliente es sencilla. Cuando se detecta un cambio en la entrada de la caja de texto, se envía un mensaje de tipo “typing” al servidor indicándole que estamos escribiendo (indicado con el valor booleano true). Después, creamos un Timeout para que, si pasa cierto tiempo sin que el usuario haya escrito, se le diga al servidor que ya no está escribiendo. Es importante que se limpie este intervalo cada vez que el usuario escriba y, además, que guardemos el estado de escribiendo para no volver a enviar un mensaje al servidor cada vez que el usuario modifique el texto de la caja de texto.

Form.on(‘input’, () => {

if (!imTyping)

{

socket.emit(‘typing’, true);

imTyping = true;

}

clearTimeout(typingTimeout);

typingTimeout = setTimeout(() => {

socket.emit(‘typing’, false);

imTyping = false;

}, 1000);

});

Por otro lado, el servidor recibirá el dato de tipo booleano y enviará a todos los clientes conectados un mensaje “nickname is typing…”, cuando el dato sea true y “No one is typing” en caso contrario.

socket.on("typing", (isTyping) => {

if (isTyping) {

socket.broadcast.emit(

"typing",

`${connectedUsers[socket.id]} is typing...`

);

} else {

socket.broadcast.emit("typing", `No one is typing`);

}

});

Finalmente, el cliente recibe el mensaje y lo muestra por pantalla.

Socket.on(‘typing’, msg => {

typingText.text(msg);

});

**Mostrar la lista de usuarios conectados**

Para esta tarea se ha creado una función dentro de la función .ready(). Esta función permite la activación del código javascript una vez que el árbol esté cargado, la función que se ha creado listUsers se ejecutará cada 5 segundos para comprobar que usuarios están conectados y se muestra en un listado en la esquina superior de la pantalla.

$(document).ready(function() {

function listUsers() {

socket.emit('list', '');

}

setInterval(listUsers, 5000);

});

El servidor recibe la llamada del cliente y devuelve un listado con los usuarios conectados.

socket.on('list', () => {

let msg = '[';

for (let c in connectedUsers) {

msg += connectedUsers[c] + ', ';

}

msg = msg.substring(0, msg.length -2);

msg += ']';

io.emit('list user', msg);

});

El cliente comprueba si el listado que ha recibido del servidor ha cambiado, si es así vacía el elemento del listado de los usuarios y vuelve a imprimir el nuevo listado con los usuarios.

socket.on('list user', function (msg) {

if(listUsers!=msg) {

listUsers=msg

$('#users').empty();

var array = msg.split(/[\\/[\d\]]/);

array = array[1].split(",")

for(let value of array){

$('#users').append($('<li>').text(value));

}

}

});

**Añadir soporte para mensajes privados**

Para esta tarea se ha creado otro tipo de mensaje llamado “pm”. Cuando el usuario escribe el mensaje “/pm destUser msg”, se le envía un mensaje únicamente al usuario con nick igual a “destUser”. Para llevar a cabo esta tarea, cuando el usuario indica al sistema que quiere enviar el mensaje escrito, el cliente comprueba que es del tipo “pm”. Si es así, recoge el usuario destino y el mensaje, los mete en un objeto de javascript y se lo envía al servidor. Además, como no espera respuesta, añade directamente en la ventana del chat un mensaje que le indica qué mensaje se le ha enviado a quién.

else if (input.startsWith('/pm'))

{

let dest = input.split(' ')[1];

if (dest != '')

{

socket.emit('pm', {msg:input.split(' ').slice(2).join(' '),dest:dest});

$('#messages').append($('<li>').text(`Yo (privado para ${dest}): ${input.split(' ').slice(2).join(' ')}`));

}

}

Cuando el servidor recibe ese mensaje, busca en la lista de usuarios conectados aquel que coincida con el nick que recibe del cliente. Cuando lo encuentra, le envía un mensaje del tipo “chat message” que ya conocemos.

socket.on("pm", ({ msg, dest }) => {

let dest\_socket = null;

for (let c in connectedUsers) {

if (connectedUsers[c] == dest) {

dest\_socket = c;

}

}

if (dest\_socket) {

socket

.to(dest\_socket)

.emit("chat message", `${connectedUsers[socket.id]}: ${msg}`);

}

});