Práctica 4 - Segunda Parte
Servidor IoT

Noelia Escalera Mejías

29 de mayo de 2020

1. Explicación del método de comunicación

Para comunicar el servidor con 'vista_usuario.html' y con 'sensores.html' lo hemos hecho a través de socket.io. En el servidor tenemos:

```
var httpServer = http.createServer(
    function(request, response) {
        var uri = url.parse(request.url).pathname;
        if (uri=="/") uri = "/vista_usuario.html";
        else if (uri.includes('sensores')) uri = "/sensores.html";
        console.log(uri);
        var fname = path.join(process.cwd(), uri);
        fs.exists(fname, function(exists){
            if (exists) {
                fs.readFile(fname, function(err, data) {
                    if (!err) {
                        var extension = path.extname(fname).split(".")[1];
                        var mimeType = mimeTypes[extension];
                        response.writeHead(200, mimeType);
                        response.write(data);
                        response.end();
                    }
                    else {
                        response.writeHead(200, {"Content-Type": "text/plain"});
                        response.write('Error de lectura en el fichero: '+uri);
                        response.end();
                    }
                });
            }
            else{
                console.log("Peticion invalida"+uri);
                response.writeHead(200, {"Content-Type": "text/plain"});
                response.write('404 Not Found\n');
                response.end();
            }
        });
    }
);
httpServer.listen(8081);
var io = socketio.listen(httpServer);
```

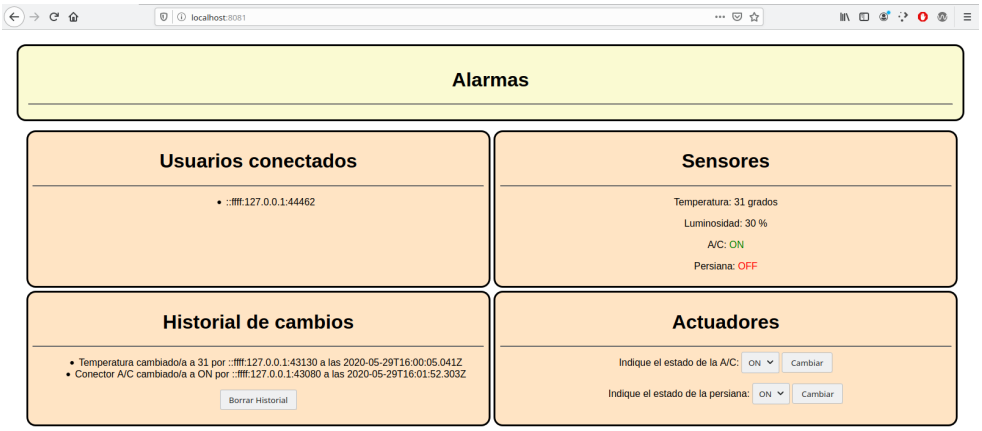
Creamos un servidor http y redireccionamos a la página que nos convenga. Luego en los clientes ya nos conectamos de la siguiente manera:

```
<script src="/socket.io/socket.io.js"></script>
<script type="text/javascript">
    var serviceURL = document.URL;
    var socket = io.connect('http://localhost:8081');
```

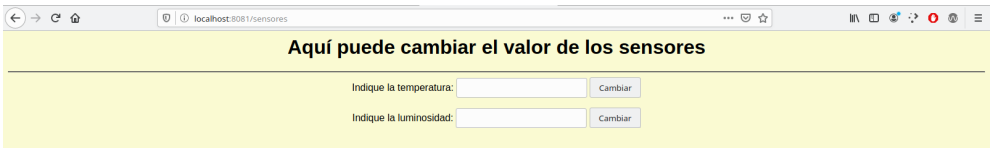
Nos conectamos al puerto 8081, ya que el 8080 dio problemas. Básicamente para cooperar entre nodos, se ha hecho uso de los emit y de los on. Cuando es necesario que emitan todos los sockets, ser hace un io.sockets en vez de que lo emita solo el cliente. Esto se va a ver más claramente a continuación en la explicación de los métodos.

2. Explicación del funcionamiento de la aplicación y de sus métodos

Al entrar a la aplicación, nos encontramos esto:



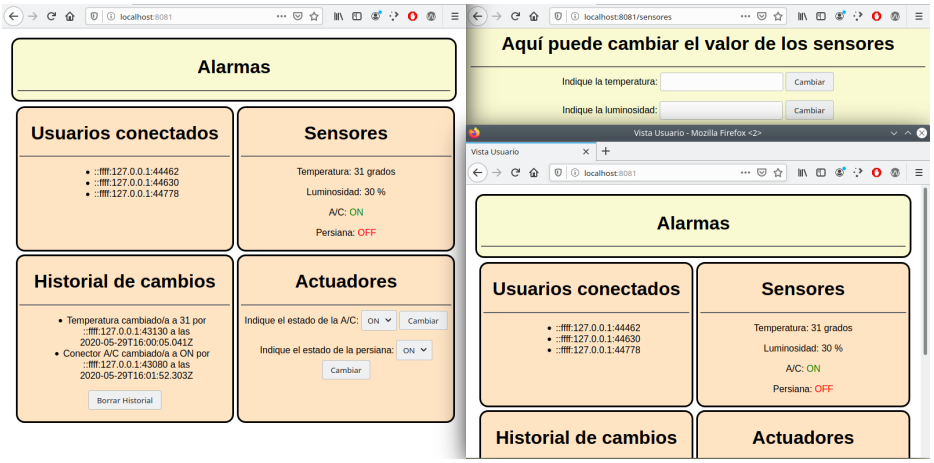
Y si nos conectamos al formulario:



Vamos a ir explicando los módulos uno por uno:

2.1. Usuarios conectados

Muestra los usuarios conectados al sistema en tiempo real, también cuentan los que estén conectados al formulario:



Esto está directamente basado en el Ejemplo 4 del guión. Aquí tenemos el método del servidor, cada vez que un usuario se conecta, lo comunica por consola al igual que cuando se desconecta:

```
var allClients = new Array();
io.sockets.on('connection', function(client) {
    allClients.push({address:client.request.connection.remoteAddress, port:client.request.connection.remotePort});
    console.log('New connection from ' + client.request.connection.remoteAddress + ':' + client.request.connection.remotePort);
    io.sockets.emit('all-connections', allClients);
    client.on('output-evt', function (data) {
        client.emit('output-evt', 'Hola Cliente!');
    });
    client.on('disconnect', function() {
        console.log("El cliente "+client.request.connection.remoteAddress+" se va a desconectar");
        console.log(allClients);

        var index = -1;
        for (var i = 0; i<allClients.length; i++){
            //console.log("Hay "+allClients[i].port);
            if(allClients[i].address == client.request.connection.remoteAddress
                && allClients[i].port == client.request.connection.remotePort){
                    index = i;
                }
        }

        if (index != -1) {
            allClients.splice(index, 1);
            io.sockets.emit('all-connections', allClients);
        } else{
            console.log("EL USUARIO NO SE HA ENCONTRADO!")
        }
        console.log('El usuario '+client.request.connection.remoteAddress+' se ha desconectado');
    });
});

console.log("Servicio Socket.io iniciado");
```

Este método emite además una llamada a 'all-connections', que se recoge en el cliente:

```
socket.on('all-connections', function(data){
    actualizarLista(data);
});
```

Como vemos, se llama a la función actualizarLista, veamos su código:

```
function actualizarLista (usuarios){
    var listContainer = document.getElementById('lista_usuarios');
    listContainer.innerHTML  = '';
    var listElement = document.createElement('ul');
    listContainer.appendChild(listElement);
    var num = usuarios.length;
    for (var i=0; i<num; i++){
        var listItem = document.createElement('li');
        listItem.innerHTML = usuarios[i].address+":"+usuarios[i].port;
        listElement.appendChild(listItem);
    }
}
```

Esta función recorre el array de usuarios y va imprimiendo su dirección y su puerto en el sitio correspondiente.

2.2. Sensores y actuadores

El campo sensores muestra el estado actual de todas las variables del sistema en tiempo real. El campo actuadores permite accionar el A/C y la persiana. Podemos modificar el valor de la luminosidad y la temperatura mediante el formulario. He aquí un ejemplo:

Veamos como se consigue esto. Básicamente tenemos una colección de MongoDB para cada variable. Cada vez que modifiquemos el valor de uno de los sensores, se insertará una nueva fila en la colección del sensor. Como los métodos son prácticamente iguales en las cuatro variables vamos a ver los del A/C como ejemplo de actuador y los de Temperatura como ejemplo de sensor. Empecemos con los de A/C:

```
dbo.createCollection("valor_ac", function(err, collection){
    io.sockets.on('connection',
    function(client){
        client.emit('inicializar_ac', collection.find({}).toArray(function(err, results) {
            client.emit('init_ac', results);
        }));
        client.on('add_nueva_ac', function(data){
            io.sockets.emit('update_ac', {ac: data.ac});
            collection.insert(data, {safe:true}, function(err, result) {});
        });
        client.on('borrar', function(data){
            collection.remove({});
            collection.insert({ac: data.ac}, {safe:true}, function(err, result){});
        });
    });
});
```

Vamos a obviar por ahora 'borrar', que lo explicaremos con el historial de cambios. Nada más conectarse un cliente, se hará un emit de 'inicializar_ac', que recogerá todas los datos de la colección y a continuación hará un emit de 'init_ac'. Este emit será recogido en el cliente:

```
socket.on('init_ac', function(data){
    inicializarAC(data);
});
```

Como vemos, simplemente llama a la funcion 'inicializarAC', a la que le pasa

los datos recogidos de la consulta, veamos qué hace esta función:

```
function inicializarAC(data) {
    var element = document.getElementById('ac');
    var num = data.length - 1;
    if (num>0)
        element.innerHTML = data[num].ac;
    else
        element.innerHTML = "OFF";

    if(document.getElementById('ac').innerHTML == "OFF"){
        document.getElementById('ac').style.color = "red";
    }
    else{
        document.getElementById('ac').style.color = "green";
    }
}
```

Primero va a comprobar que haya algún valor anteriormente, si no pondrá por defecto a 'OFF', sino, pues pondrá el valor de la última fila insertada, ya que será el valor más reciente que tuvo la variable antes de apagarse el sistema. También pondrá el texto en rojo o verde, dependiendo de si está encendido o apagado.

En los métodos del servidor veíamos un client.on('add_nueva_ac'):

```
client.on('add_nueva_ac', function(data){
    io.sockets.emit('update_ac', {ac: data.ac});
    collection.insert(data, {safe:true}, function(err, result) {});
});
```

Podemos ver que hace un emit a todos los sockets de 'update_ac' y luego inserta los datos que le pasamos a la colección. ¿Pero desde dónde se llama a esta función? Pues se llama cuando pulsamos cambiar en el formulario del A/C:

```
<form action="javascript:void(0);" onsubmit="javascript:ONOFFAC();">
    <label for="ac_label">Indique el estado de la A/C: </label>
    <select id="ac_elegir">
        <option value="ON">ON</option>
        <option value="OFF">OFF</option>
    </select>
    <input type="submit" value="Cambiar">
</form>
```

```
function ONOFFAC() {
    var d = new Date();
    var on_off = document.getElementById('ac_elegir').value;
    var variable = "Conector A/C";
    socket.emit('add_cambio', {variable: variable, value: on_off, date:d});
    socket.emit('add_nueva_ac', {ac:on_off});
}
```

Al enviar el formulario, llamamos a la función 'ONOFFAC()'. Esta función es la que hace el emit de 'add_nueva_ac' y le manda el nuevo valor del A/C, metiéndolo en la colección desde el servidor, además hace un emit de 'add_cambio', que luego explicaremos en detalle, pero adelantamos que tiene que ver con el historial.

Vamos a ver ahora los métodos de Temperatura:

```
dbo.createCollection("valor_temperatura", function(err, collection){
    io.sockets.on('connection',
    function(client){
        client.emit('inicializar_temperatura', collection.find({}).toArray(function(err, results){
            client.emit('init_temperatura', results);
        }));
        client.on('add_nueva_temperatura', function(data){
            io.sockets.emit('update_temperatura', {temperatura: data.temperatura});
            collection.insert(data, {safe:true}, function(err, result) {});
        });
        client.on('borrar', function(data){
            collection.remove({});
            collection.insert({temperatura: data.temperatura}, {safe:true}, function(err, result){});
        });
    });
});
```

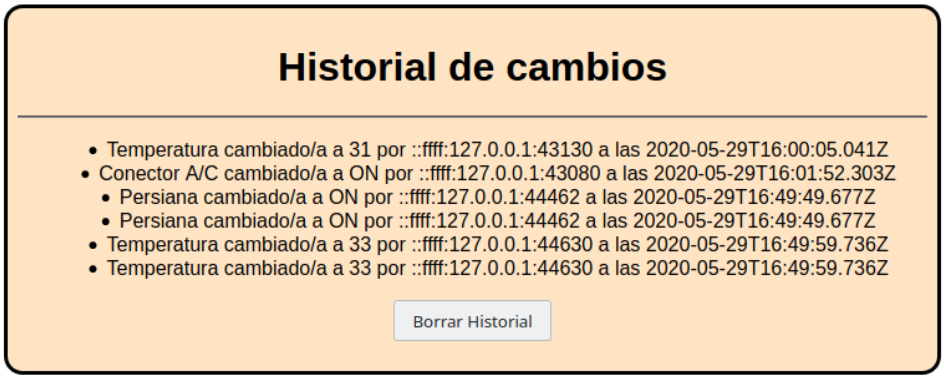
Podemos observar que son los mismos que los de A/C, la diferencia es que la función que llama en este caso a 'add_nueva_temperatura' está en el archivo de 'sensores.html', ya que se llama desde el formulario que hay ahí:

```
<form action="javascript:void(0);" onsubmit="javascript:cambiarTemperatura();">
    <label for="temperatura">Indique la temperatura: </label>
    <input type="text" id="temperatura_valor"/>
    <input type="submit" value="Cambiar">
</form>
```

```
function cambiarTemperatura(){
    var d = new Date();
    var temp = document.getElementById('temperatura_valor').value;
    var variable = "Temperatura";
    socket.emit('add_cambio', {variable: variable, value: temp, date:d});
    socket.emit('add_nueva_temperatura', {temperatura:temp});
}
```

2.3. Historial de cambios

El historial de cambios recoge todos los cambios que se han hecho en las medidas, el usuario que las realizó y la fecha:



Tenemos una colección que almacena todos estos datos (variable cambiada, nuevo valor, usuario y fecha). Esto es lo que tenemos en el servidor:

```
dbo.createCollection("historico_sensores", function(err, collection){
    io.sockets.on('connection',
    function(client){
        client.emit('inicializar', collection.find({}).toArray(function(err, results){
            client.emit('obtener',results);
        }));
        client.on('borrar', function(data){
            collection.remove({});
        });
        client.on('add_cambio', function (data) {
            io.sockets.emit('update_cambios', {variable:data.variable, value:data.value, host:client.request.connection.remoteAddress, port:client.request.c
        });
        client.on('poner', function(data){
            collection.insert(data, {safe:true}, function(err, result) {});
        });
        client.on('obtener', function() {
            collection.find({}).toArray(function(err, results){
                client.emit('obtener', results);
            });
        });
        client.on('apagar_persiana', function(){
            var d = new Date();
            var data = {};
            data.variable = "Persiana";
            data.value = "OFF";
            data.host = "Agente";
            data.port = "Agente";
            data.date = d;
            io.sockets.emit('update_cambios', {variable:data.variable, value:data.value, host:data.host, port:data.port, date:data.date});
        });
    });
});
```

Nada más conectarse un cliente se hace un emit de 'inicializar', que carga todo el historial, y se hace un emit de 'obtener', al que se le pasan estos datos (como vemos es muy similar a lo de los sensores). El emit de 'obtener' se captura en el cliente:

```
socket.on('obtener', function(data){
    actualizarCambios(data);
});
```

Esto llama a la función actualizarCambios:

```
function actualizarCambios(data){
    var listContainer = document.getElementById('cambios');
    listContainer.innerHTML = '';
    var listElement = document.createElement('ul');
    listContainer.appendChild(listElement);
    var num = data.length;
    for (var i=0; i<num; i++) {
        var listItem = document.createElement('li');
        listItem.innerHTML = data[i].variable + " cambiado/a a " + data[i].value + " por " + data[i].host + ":" + data[i].port + " a las " + data[i].date;
        listElement.appendChild(listItem);
    }
    if (document.getElementById('pers').innerHTML == "OFF" && document.getElementById('ac').innerHTML == "OFF"){
        document.getElementById('persianayac').innerHTML = "Atención, persiana y A/C apagados";
    }
    else{
        document.getElementById('persianayac').innerHTML = "";
    }
}
```

Básicamente recorre todos los cambios y los pone en el sitio indicado. También lanza una alarma si la persiana y el aire acondicionado están apagados. Veamos ahora qué es lo que hace el método 'borrar', que lo tenemos tanto aquí como en los sensores. Este método se llama cuando pulsamos el botón 'Borrar Historial':

```
<div id="cambios"></div>
<form action="javascript:void(0);" onsubmit="javascript:borrarHistorial();">
<input type="submit" value="Borrar Historial"/>
```

```
function borrarHistorial(){
    data = {};
    data.ac = document.getElementById('ac').innerHTML;
    data.persiana = document.getElementById('pers').innerHTML;
    data.temperatura = document.getElementById('tem').innerHTML;
    data.luminosidad = document.getElementById('lum').innerHTML;
    socket.emit('borrar', data);
    document.getElementById('cambios').innerHTML = "";
}
```

Como vemos, guarda los valores actuales de las variables, hace un emit de borrar y deja el espacio del historial en blanco. Veamos lo que hacen los distintos borrar del servidor:

```
client.on('borrar', function(data){
    collection.remove({});
});
```

En la colección de historial, simplemente borra todos los datos.

```
client.on('borrar', function(data){
    collection.remove({});
    collection.insert({ac: data.ac}, {safe:true}, function(err, result){});
});
```

En las colecciones de las variables, borra también los datos, pero vuelve a meter el último valor almacenado, que lo hemos pasado desde el cliente, esto nos sirve para que siempre quede un valor recordado.

Vamos a ver ahora el método 'add_cambio':

```
client.on('add_cambio', function (data) {
    io.sockets.emit('update_cambios', {variable:data.variable, value:data.value, host:client.request.connection.remoteAddress, port:client.request.connection.
});
```

'add_cambio' ya vimos que se emitía cuando se cambiaba una variable, y le pasábamos la variable cambiada, su nuevo valor y la fecha. Al recibir este mensaje, todos los sockets emiten 'update_cambios', al cual se le pasan todos estos datos, además de la ip y del puerto. Este método se escucha en el cliente:

```
socket.on('update_cambios', function(data){
    socket.emit('poner', data);
    socket.emit('obtener', data);
});
```

Esta función hace emit de 'poner' y de 'obtener'. Ambos métodos se encuentran en el servidor:

```
client.on('poner', function(data){
    collection.insert(data, {safe:true}, function(err, result) {});
});
client.on('obtener', function() {
    collection.find({}).toArray(function(err, results){
        client.emit('obten', results);
    });
});
```

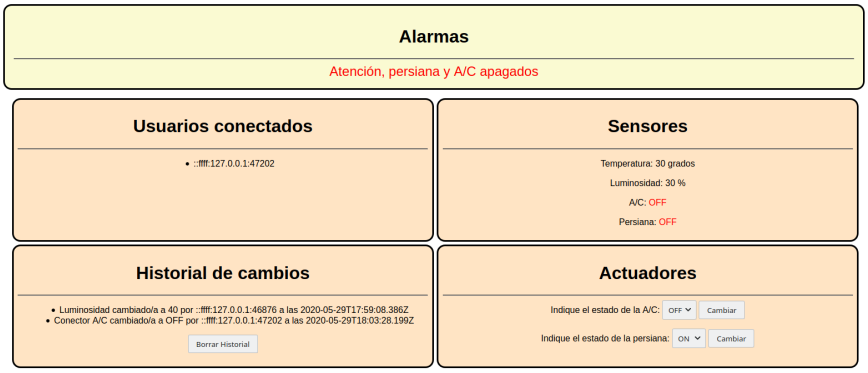
'poner', inserta en la colección del historial el nuevo cambio. 'obtener' obtiene todos los cambios hasta ahora y emite 'obten', en este caso para el cliente:

```
socket.on('obten', function(data){
    actualizarCambios(data);
});
```

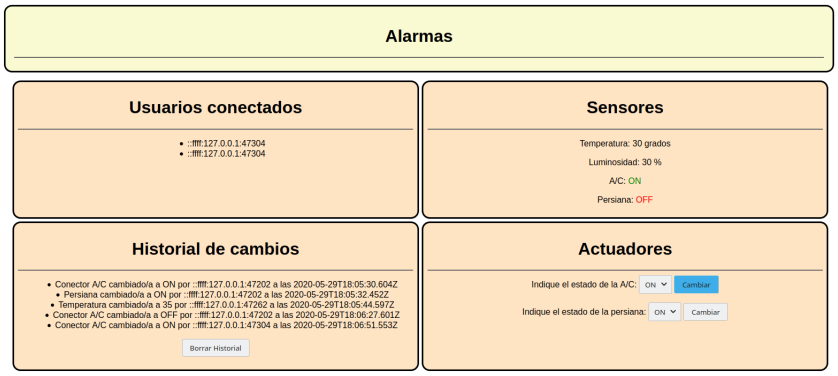
Como vemos, este método simplemente llama a 'actualizarCambios' (función que ya hemos visto) para actualizar el historial.

2.4. Alertas

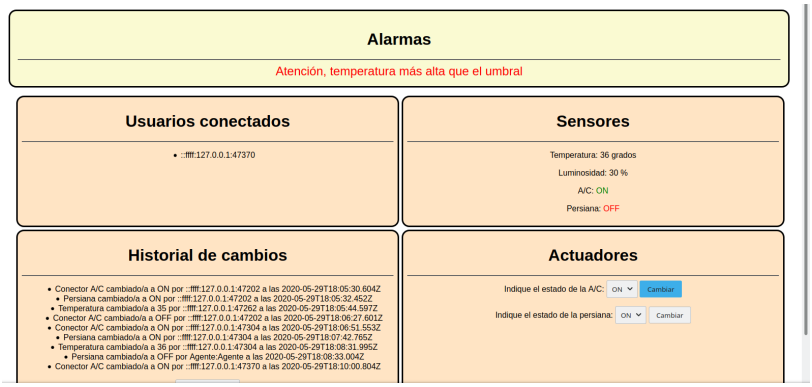
En esta sección se nos muestran las alertas del sistema, ya hemos visto una, que se activa cuando están apagadas tanto la persiana como el A/C y que se comprueba cuando se actualizan los cambios:



Cuando una de las dos se enciende, la alarma cesa:



Las otras alarmas que hay son cuando bien la temperatura, bien la luminosidad sobrepasan el límite (además la persiana se apaga automáticamente, y en el historial se recoge que lo ha hecho el agente):



Vamos a explicar esta alarma con la temperatura, con la luminosidad es igual.

```
function actualizarTemperatura(data){
    var element = document.getElementById('tem');
    element.innerHTML = data.temperatura;
    if (data.temperatura>35){
        document.getElementById('alarma_tem').innerHTML = "Atención, temperatura más alta que el umbral";
        var off = "OFF";
        socket.emit('apagar_persiana', {persiana:off});
    }
    else if (data.temperatura<20){
        document.getElementById('alarma_tem').innerHTML = "Atención, temperatura más baja que el umbral";
    }
    else{
        document.getElementById('alarma_tem').innerHTML = "";
    }
}
```

```
function inicializarTemperatura(data){
    var element = document.getElementById('tem');
    var num = data.length - 1;
    if (num>0){
        element.innerHTML = data[num].temperatura;
        if(data[num].temperatura > 35){
            document.getElementById('alarma_tem').innerHTML = "Atención, temperatura más alta que el umbral";
        }
        else if (data[num].temperatura<20){
            document.getElementById('alarma_tem').innerHTML = "Atención, temperatura más baja que el umbral";
        }
        else{
            document.getElementById('alarma_tem').innerHTML = "";
        }
    }
    else
        element.innerHTML = "30";
}
```

Cuando bien inicializamos el sistema, o bien cuando actualizamos la temperatura/luminosidad, hay que comprobar si alguno de estos valores está por debajo o por encima del umbral. Si es el caso, se lanzará una alarma. Si se ha superado el umbral alto, además se hará un emit de 'apagar_persiana'. Esto solo

se hace en actualizar, ya que al inicializar se supone que ya ha habido un cambio anterior y por tanto ya se ha bajado la persiana. 'apagar_persiana' lo tenemos en el servidor, en la parte del historial:

```
client.on('apagar_persiana', function(){
    var d = new Date();
    var data = {};
    data.variable = "Persiana";
    data.value = "OFF";
    data.host = "Agente";
    data.port = "Agente";
    data.date = d;
    io.sockets.emit('update_cambios', {variable:data.variable, value:data.value, host:data.host, port:data.port, date:data.date});
});
```

Básicamente mandamos un cambio al historial, esta vez hecho por el agente. En los métodos de la colección de Persiana, también escuchamos 'apagar_persiana':

```
client.on('apagar_persiana', function(data){
    io.sockets.emit('persiana_off', data);
    collection.insert(data, {safe:true}, function(err, result) {});
});
```

Aquí hacemos que todos los sockets emitan 'persiana_off' y se inserta en la colección de persiana el nuevo cambio. 'persiana_off' tiene su socket.on en el cliente:

```
socket.on('persiana_off', function(data){
    apagarPersiana();
});
```

```
function apagarPersiana(){
    var element = document.getElementById('pers');
    element.innerHTML = "OFF";
    element.style.color = "red";
}
```

Este método llama a 'apagarPersiana', que informa a los usuarios del cierre de la persiana.