ugr | Universidad de Granada

Desarrollo de Sistemas Distribuidos

Práctica 4: Nodejs

José Manuel Navarro Cuartero

Índice:

Ejemplos

helloworld.js

Este ejemplo es el más simple que trataremos. Se obtiene el módulo http de node, y se crea el servidor con dos parámetros. En el objeto “request” se codifica el mensaje de petición recibido en el servicio. Con el objeto “response” podemos crear una respuesta a la petición recibida, como se hace en este caso, escribiendo en la web. Por último, vemos que el servidor estará en escucha en el puerto 8080.

```
var http = require("http");
var httpServer = http.createServer(
    function(request, response) {
        console.log(request.headers);
        response.writeHead(200, {"Content-Type": "text/plain"});
        response.write("Hola holita, mundo");
        response.end();
    }
);
httpServer.listen(8080);
console.log("Servicio HTTP iniciado");
```

calculadora.js

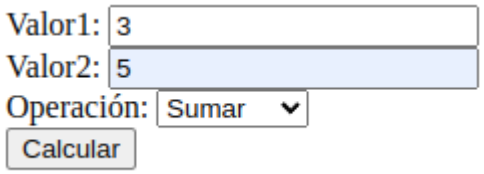
En este ejemplo tenemos la función principal “createServer” en la que iniciaremos el servidor en localhost, que posteriormente pondremos a escuchar en el puerto 8080. El servidor tomará los argumentos para el cálculo de la url de la propia web.
Así pues, para hacer cualquier cálculo haremos “localhost:8080/sumar/3/5”, el servidor se ocupará de realizar el cálculo llamando a una función “calcular” propia, y nos sacará por pantalla en la web el resultado.

```
// se separan los elementos sin las /
var params = uri.split("/");
if (params.length == 3) {
    var val1 = parseFloat(params[1]);
    var val2 = parseFloat(params[2]);
    var result = calcular(params[0], val1, val2);
    output = result.toString();
}
else output = "Error: El número de parámetros no es válido";

response.writeHead(200, {"Content-Type": "text/html"});
response.write(output);
response.end();
```

calculadora-web.js

Este ejemplo será una “ampliación” del anterior. El servidor ahora proporcionará una interfaz web para los cálculos, y así, en lugar de recibir los cálculos a realizar a través de la URL, lo hará a través de la interfaz web.



Se obtienen los valores introducidos por el usuario en el formulario, y se crea una petición tipo REST, y finalmente se envía una petición al servicio de manera asíncrona con XMLHttpRequest.

connections.js

En este ejemplo introducimos los eventos, un elemento de vital importancia para la práctica. De nuevo tendremos por un lado el servidor, y por otro la interfaz “connections.html”.
Cada vez que se conecta un nuevo cliente se llama al evento io.sockets.on ('connection', function(client){}), que ejecutará todo lo que se halle entre los corchetes, teniendo a mano el argumento client. En este caso, se guardan los datos del cliente en un array, y se transmiten esos datos a todos los clientes con el evento “all-connections”.

```
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!');
});
```

De forma análoga, en una desconexión también se tendrá en cuenta y se avisará.

```
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);
```

El html simplemente mostrará la lista de conectados.

mongo-test.js

Aquí usaremos una base de datos no-SQL, MongoDB, a la cual nos conectaremos accediendo al puerto que utiliza, 27017.

```
MongoClient.connect("mongodb://localhost:27017/",
    { useUnifiedTopology: true }, function(err, db) {
    httpServer.listen(8080);
    var io = socketio(httpServer);
    var dbo = db.db("pruebaBaseDatos");
```

De nuevo haremos uso de los eventos para intercambiar datos entre la interfaz web, y el servidor. El servidor es el que tendrá acceso a la base de datos para insertar tuplas o hacer búsquedas.

```
io.sockets.on('connection', function(client) {
    // Se le pasa al cliente su propia dirección
    client.emit('my-address', {host:client.request.connection.remoteAddress,
        port:client.request.connection.remotePort});
    // El cliente la manda de vuelta para que se guarde
    client.on('poner', function (data) {
        collection.insertOne(data, {safe:true}, function(err, result) {});
    });
    // El cliente pide que se le envíen los datos guardados
    client.on('obtener', function (data) {
        collection.find(data).toArray(function(err, results){
            client.emit('obtener', results);
        });
    });
});
```

El cliente por su lado, se encargará de hacer peticiones y recibir respuestas.

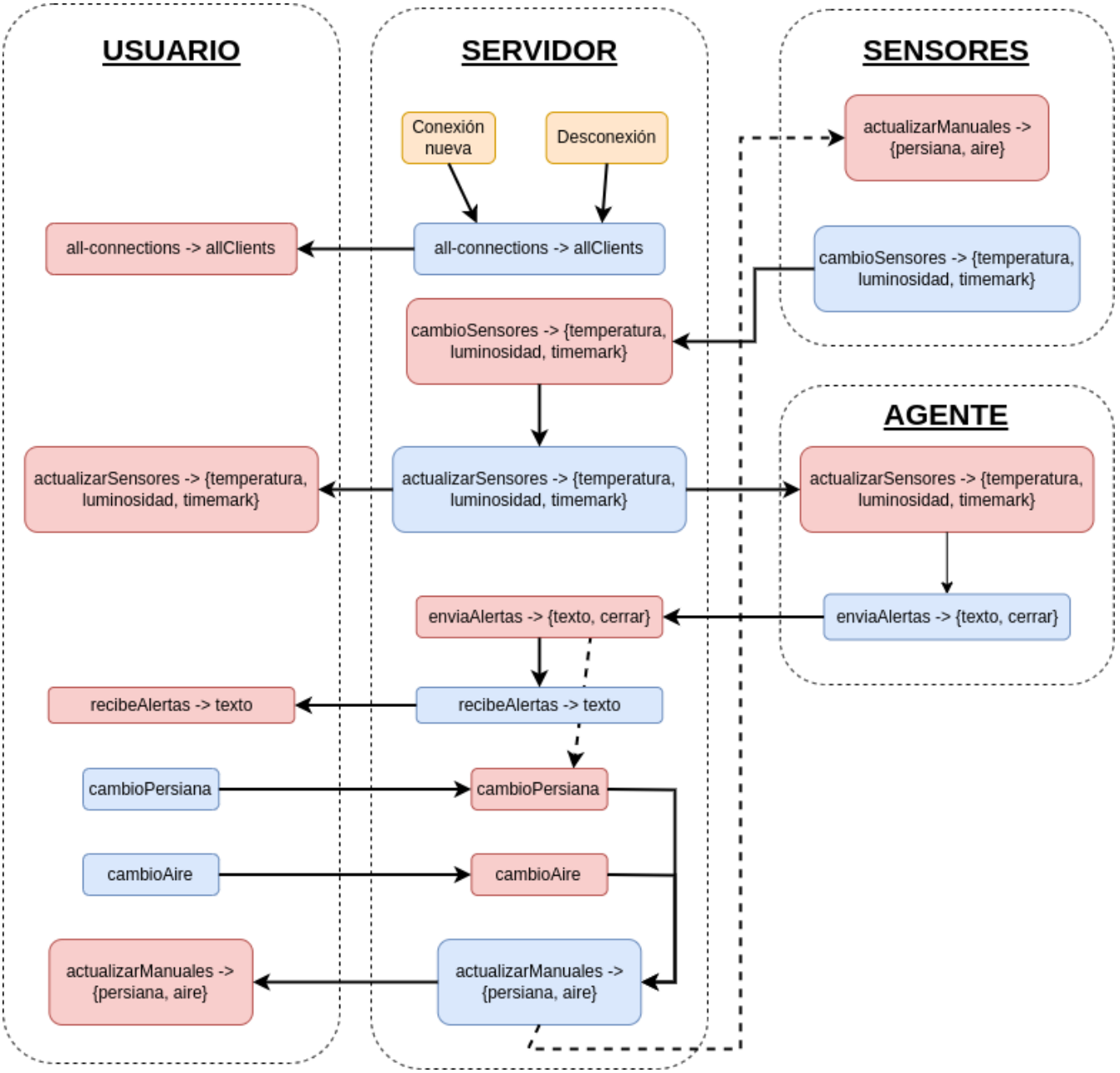
```
socket.on('my-address', function(data) {
    var d = new Date();
    socket.emit('poner', {host:data.host, port:data.port, time:d});
    socket.emit('obtener', {host: data.address});
});
socket.on('obtener', function(data) {
    actualizarLista(data);
});
// En desconexion, se queda la pantalla blanca (porque no mestra conexiones)
socket.on('disconnect', function() {
    actualizarLista({});
});
```

Ejercicio

Nuestro ejercicio se compondrá del servidor y 3 vistas, la de usuario, la de los sensores y la del agente. Las 3 vistas enviarán eventos al servidor, y éste les responderá. Además, el servidor accederá a la base de datos de mongo.

Conexiones

En este diagrama, los cuadrados azules representan eventos de envíos (emit) y los rojos de recepciones (on), de forma “evento -> argumento”.



Servidor

El servidor tendrá varias funciones. Llevará el recuento de los clientes conectados (de forma análoga al ejemplo de connections), almacenará en una base de datos los datos que reciba de los sensores (análogo al ejemplo mongo-test), creará el servidor para que escuche peticiones en el puerto 8080 y además hará de intermediario entre las distintas vistas, respondiendo a sus eventos.
Por ejemplo, cuando los sensores envían nuevas medidas, se registran y se envían a los clientes.

```
// Recibimos nuevas medidas de la interfaz de los sensores y las enviamos a los usuarios
client.on('cambioSensores', function(data){
    datosSensores = data;
    console.log(datosSensores);
    collection.insertOne(datosSensores, {safe:true}, function(err, result) {});
    //console.log(collection.find().toArray());
    io.sockets.emit('actualizarSensores',datosSensores);
});
```

Si es el cliente el que modifica el aire o la persiana, también se registra y envía.

```
// Cuando el cliente cambia el aire o la persiana, se lo notifica al servidor,
// que se lo traspasa a los clientes despues
client.on('cambioAire', function(){
    aireEncendido = !aireEncendido;
    io.sockets.emit('actualizarManuales',{persiana: persianaSubida, aire: aireEncendido});
});
client.on('cambioPersiana', function(){
    persianaSubida = !persianaSubida;
    io.sockets.emit('actualizarManuales',{persiana: persianaSubida, aire: aireEncendido});
});
```

Y si el agente envía una alerta.

```
// Se recibe una alerta del agente y se envia a los clientes
client.on('enviaAlertas',function(data){
    console.log(data.texto);
    // Si se nos pide que se cierre/baje la persiana y está subida, la bajamos
    if(data.cerrar && persianaSubida){
        console.log("Bajamos la persiana a peticion del agente");
        persianaSubida = false;
        io.sockets.emit('actualizarManuales',{persiana: persianaSubida, aire: aireEncendido});
    }
    io.sockets.emit('recibeAlertas',data.texto);
});
```

Además, en cada nueva conexión, se proporcionan los datos que se manejan en ese instante, tanto de temperatura y luminosidad, como del aire acondicionado y la persiana.

```
// Notificamos a los clientes nuevos las conexiones y el estado de persiana y aire
io.sockets.emit('all-connections', allClients);
io.sockets.emit('actualizarManuales',{persiana: persianaSubida, aire: aireEncendido});
io.sockets.emit('actualizarSensores',datosSensores);
```

Usuario

La vista de usuario simplemente recopilará la información disponible en pantalla, además de permitir a los clientes encender o apagar el aire acondicionado, abrir y cerrar la persiana. También sacará por pantalla los mensajes de alerta que reciba.
Principalmente recibirá por un lado un objeto data con la temperatura, la luminosidad y el momento de la meidición de los sensores.

```
// Se actualiza la info de los sensores
socket.on('actualizarSensores', function(data){
    console.log("Llega info de los sensores");
    console.log(data);
    const temperatura = document.getElementById("temperatura");
    const luminosidad = document.getElementById("luminosidad");
    const fecha = document.getElementById("timemark");
    temperatura.innerText = (data.temperatura) ? data.temperatura : "No definido todavia";
    luminosidad.innerText = (data.luminosidad) ? data.luminosidad : "No definido todavia";
    fecha.innerText = (data.timemark) ? data.timemark : "No definido todavia";
});
```

Y por otro lado también podrá recibir el estado del aire acondicionado y la persiana.

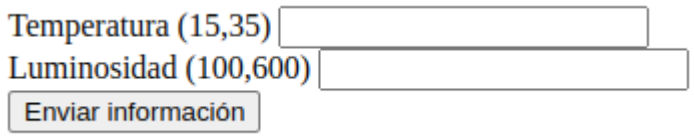
```
// Se actualizan persiana y aire con la info del servidor
socket.on('actualizarManuales', function(data){
    console.log("Llega info de los manuales...");
    console.log(data);
    const aire = document.getElementById("aire");
    const persiana = document.getElementById("persiana");
    (data.aire == true) ? aire.innerText = "Encendido" : aire.innerText = "Apagado";
    (data.persiana == true) ? persiana.innerText = "Subida" : persiana.innerText = "Bajada";
});
```

Para la interacción con aire acondicionado y persiana, el html tendrá dos botones que llamarán a funciones que a su vez harán un “emit” al servidor.

```
// Funciones de los dos botones
function changeAC(){
    socket.emit('cambioAire');
}
function changePersiana(){
    socket.emit('cambioPersiana');
}
```

Sensores

La interfaz de los sensores permitirá introducir manualmente los datos de temperatura y luminosidad.



Cuando se pulse el botón se llamará a una función que se encargará de recoger esos datos y enviarlos al servidor.

```
function sendInfo(){
    const temp = document.getElementById("temperatura").value;
    const lumi = document.getElementById("luminosidad").value;
    const date = new Date();
    const time = date.toString();

    socket.emit('cambioSensores', {temperatura:temp, luminosidad:lumi, timemark: time});

    document.getElementById("temperatura").value = "";
    document.getElementById("luminosidad").value = "";
}
```

Agente

El agente recibirá los datos de los sensores como otro cliente más, pero cuando la luz o la luminosidad se salgan de unos márgenes preestablecidos dará la voz de alarma al servidor, para que éste la reenvíe a los clientes. Además, en algunos casos, dará la orden de que se cierre la persiana automáticamente.

```
if (data.luminosidad < LMin) {
    string +='¡Alerta! Poca luz\n';
}
if (data.luminosidad > LMax) {
    string +='¡Alerta! Mucha luz\n';
    cerrarPersiana = true;
}
console.log(string);
socket.emit('enviaAlertas',{texto:string, cerrar:cerrarPersiana});
```

Uso

Para utilizar el sistema, simplemente ejecute la orden “nodejs servidor.js” y en el navegador abra las distintas interfaces, la de usuario en “localhost:8080”, y las de agente y sensores en “localhost:8080/agente.html” y “localhost:8080/sensores.html”, respectivamente.