Práctica 4 : Node.js

Diseño de Sistemas Distribuidos

Alba Guisado Farnes

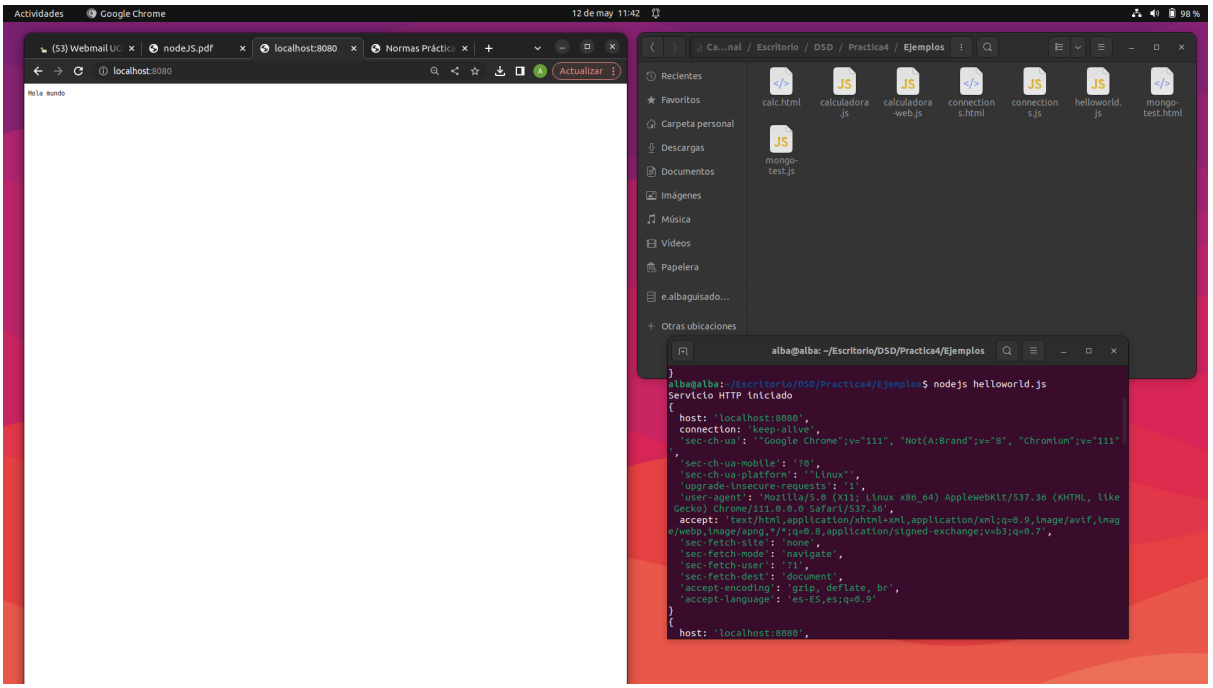
1. Implementación de ejemplos

Implementaremos unos ejemplos previamente a la realización de nuestro ejercicio para entender correctamente el funcionamiento de node.js, socket.io y mongoDB.

1.1.Helloworld

Este primer ejemplo es un ejemplo sencillo de la utilización de node.js dónde únicamente mostramos un “Hola mundo” en el navegador. Para ejecutarlo lo lanzamos con el siguiente comando:

```
nodejs helloworld.js
```



1.2.Calculadora.js

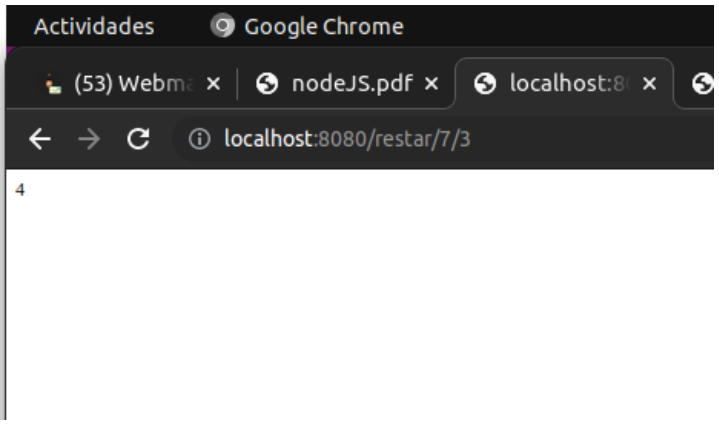
En este ejemplo creamos una calculadora distribuida usando una interfaz de tipo REST. Para poder realizar las operaciones es necesarias pasarlas por el navegador, un ejemplo sería:

```
http://localhost:8080/restar/7/3
```

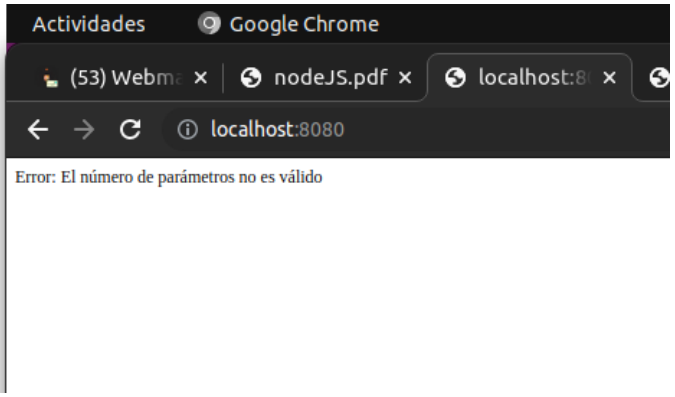
Para ello previamente debemos de ejecutar desde la terminal el ejemplo de la siguiente forma:

```
nodejs calculadora.js
```

Las operaciones implementadas son las básicas: sumar, restar, dividir y producto.



En el caso que no se pase ningún parámetro daría error:

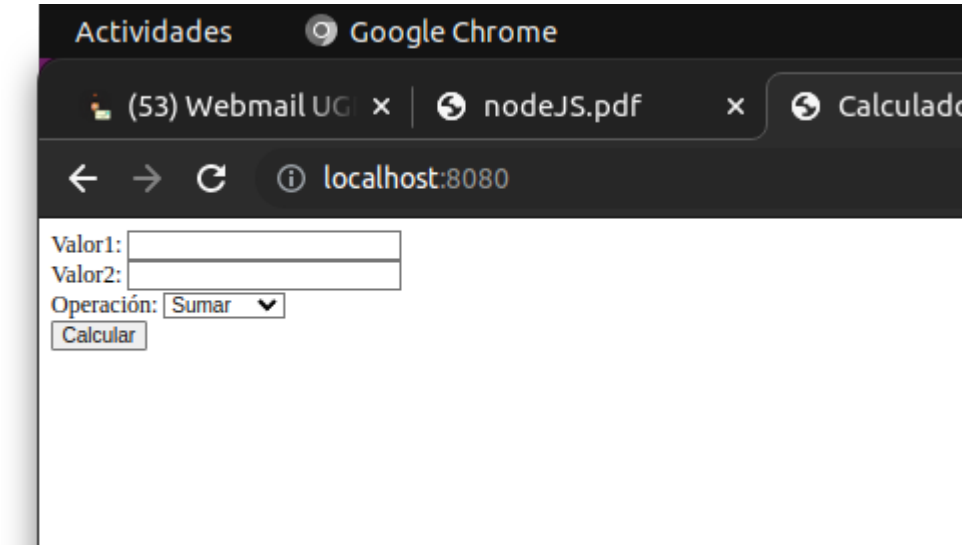


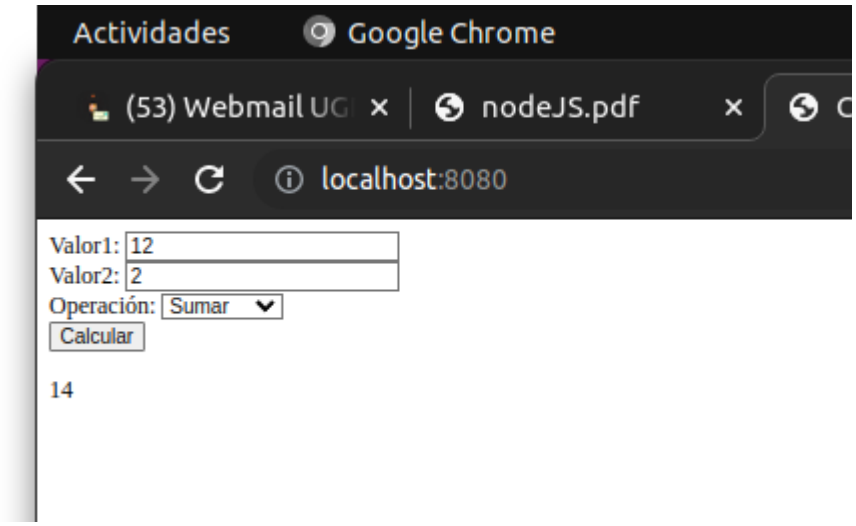
1.3. Calculadora-web.js

Este es otro ejemplo de calculadora pero en este caso no es necesario que le pasemos los parámetros en la url. Gracias al archivo calc.html se crea una interfaz gráfica dónde podemos introducir las operaciones de manera más intuitiva y sencilla mediante un formulario. Es calc.html el que se encarga de pasarle los datos a calculadora-web.js para que estos sean de tipo REST. Es necesario ejecutar el siguiente comando:

```
nodejs calculadora-web.js
```

La interfaz que nos aparece es la siguiente:





1.4. Aplicaciones en tiempo real con socket.io

Utilizaremos socket.io para dar soporte a aplicaciones en tiempo real en node.js. El siguiente ejemplo notificará las direcciones de todos los clientes que hay conectados a un servicio. Este enviará un mensaje cada vez que un cliente se conecta o desconecta al resto de usuarios conectados. Además enviará un “Hola cliente!” al usuario que se acaba de conectar. Para este ejemplo han utilizado un socket.io para realizar las peticiones *emit* que permite notificar eventos el servidor al cliente.

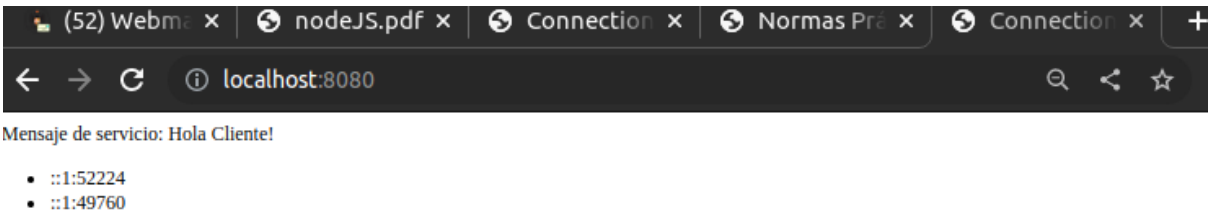
Para ejecutar el ejemplo:

```
nodejs connections.js
```

Para ver su comportamiento he realizado dos conexiones y me he desconectado de una. Desde la terminal se informa de todos los sucesos:

```
alba@alba:~/Escritorio/DSD/Practica4/Ejemplos$ nodejs connections.js
Servicio Socket.io iniciado
Peticion invalida: /favicon.ico
New connection from ::1:40090
El cliente ::1 se va a desconectar
[ { address: '::1', port: 40090 } ]
El usuario ::1 se ha desconectado
Peticion invalida: /favicon.ico
New connection from ::1:40100
Peticion invalida: /favicon.ico
New connection from ::1:40594
El cliente ::1 se va a desconectar
[ { address: '::1', port: 40100 }, { address: '::1', port: 40594 } ]
El usuario ::1 se ha desconectado
El cliente ::1 se va a desconectar
[ { address: '::1', port: 40100 } ]
El usuario ::1 se ha desconectado
Peticion invalida: /favicon.ico
New connection from ::1:52224
```

Podemos ver la salida por pantalla en el navegador:



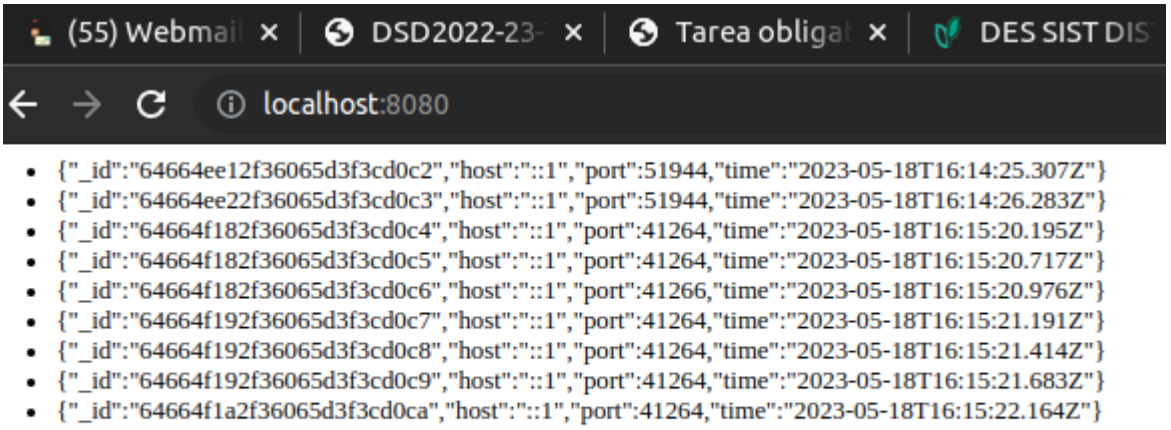
1.5. Uso de MongoDB desde Node.js

En este ejemplo nos ayudaremos de MongoDB, una base de datos de tipo NoSQL, para así guardar la información de manera no estructurada. Para ello se utilizan las notificaciones de tipo “poner” para realizar una inserción en la BD y “obtener” para realizar una consulta. Las utilizaremos para guardar los datos de los clientes, así como también se mostraŕa la dirección de conexión del cliente cuando este se conecte.

En mongo-test.js se llama socket.io dentro de la función definida por mongoClient.connect para así hacer uso de la BD. También contamos con el archivo mongo-test.html para poder mostrar los datos de los usuarios y registrar a todos ellos. Definimos todas las funciones de socket.io en el archivo javaScript para luego hacer uso de ellas en el host dónde se ejecuta el cliente.

Para la ejecución del ejemplo utilizamos el siguiente comando:

```
node mongo-test.js
```

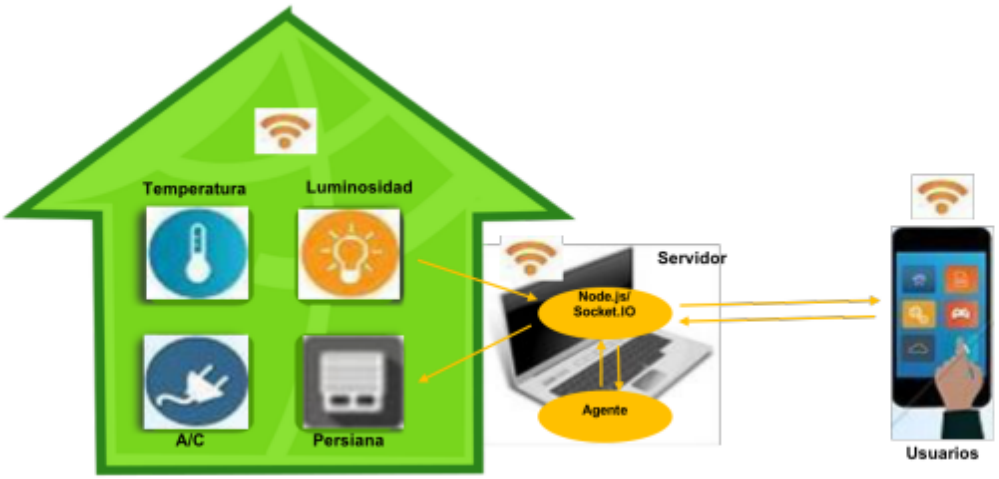


2. Sistema domótico

Se nos pide implementar un sistema domótico básico compuesto con varios sensores, varios actuadores y un servidor. Se debe comportar de la siguiente manera:

- Sensores: difunden información sobre las medidas tomadas.
- Servidor: proporciona servicio a los sensores y a la página del cliente. Además hará uso de una base de datos para almacenar un histórico de todo lo que pasa.
- Cliente: tendrá una interfaz dónde podrá ver las medidas de los sensores, las alarmas y podrá cambiar el valor de los actuadores.
- Agente: se encargará de comprobar que las medidas de los sensores se encuentra dentro de unos umbrales, por el contrario, actuará y mandará las alarmas oportunas.

Se puede ver más claro el ejercicio a través del siguiente esquema:



Para mi sistema domótico he elegido que los sensores serán temperatura, iluminación, viento y precipitaciones. Así como los actuadores serán persiana, aire acondicionado, calefacción y ventanas.

2.1. Sensores

Para que el servidor reciba las medidas de los sensores he decidido acceder a la web del tiempo en Granada y coger los datos de ahí.

La web es la siguiente:
https://www.google.com/search?q=tiempo+en+granada&oq=timepo+en+granda&aqs=chrome.1.69i57j0i10i131i433i512l3j0i10i512l5j0i10i131i433i512.3556j1j4&sourceid=chrome&ie=UTF-8

He usado la librería puppeteer que permite acceder al valor de un recurso de una página web si sabes el id o clase al que pertenece en el html. Es necesario tener la biblioteca instalada para que funcione el sistema, si no es el caso pruebe con el comando:

```
npm install puppeteer
```

Podemos ver un ejemplo de una función, todas tienen la misma estructura:

```
async function obtenerTemperatura() {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();

  await
page.goto('https://www.google.com/search?q=tiempo+en+granada&oq=timepo+en+granda&aqs=chr
ome.1.69i57j0i10i131i433i512l3j0i10i512l5j0i10i131i433i512.3556j1j4&sourceid=chrome&ie=U
TF-8');

  await page.waitForSelector('#wob_tm');
  const temperatura = await page.$eval('#wob_tm', element => element.textContent);

  await browser.close();

  return temperatura;
}
```

Primero se inicia la conexión, se manda la web que queremos y se extrae el código del selector que hemos seleccionado. Por último cerramos la conexión y devolvemos el valor deseado. Si pasamos un id hay añadir primero '#' y si es una clase '.'.

2.2. Servidor y agente

He implementado la funcionalidad del agente dentro del mismo servidor. Una vez que he establecido la conexión con socket.io y mongodb escucho varias llamadas por parte del cliente. Estas son:

- cargarDatos
- addPersiana, addAC, addCalefaccion, addVenatana

Estas últimas simplemente son para cuando el cliente decide modificar el valor de los actuadores, dentro de ellas se añade en el historial el cambio y se emite actualizarSistema, que es la función que el cliente escucha para actualizar sus datos cuando se produce un cambio.

Podemos ver un ejemplo de una de ellas:

```
client.on('addPersiana', function (data) {
```

```
        persiana = data['persiana'];
        collection.insertOne(data, {safe:true}, function(err, result) {});
        io.sockets.emit('actualizarSistema', data);
      });
```

Dentro de la llamada cargarDatos llamamos a cada uno de los sensores, si este tiene un valor diferente al almacenado significa que ha cambiado, por lo que se alamacenará en el sistema y se emitirá actualiarSistema para la actualización del cliente. Sería tipo así:

```
obtenerTemperatura()
          .then(temperatura => {
            if(temperatura != temp){
              //Almacenamos el nuevo valor
              var d = new Date();
              collection.insertOne({ temperatura , d}, {safe:true}, function(err, result) {});
              io.sockets.emit('actualizarSistema', { temperatura });
              temp = temperatura;

...
```

Una vez que tenemos el nuevo valor procedemos a gestionar como agente este, las alarmas que tengo en mi sistema domótico son las siguientes:

Para la temperatura:

- Si es mayor que el umbral altaTemp (30º), se crea una alarma que recomienda encender el aire
- Si es mayor que el umbral MAXTemp(35º) y el aire está apagado, se enciende automáticamente
- Si es menor que el umbral bajaTemp (10º), se crea una alarma que recomienda encender la calefacción
- Si es mayor que el umbral MINTemp(5º) y la calefacción está apagado, se enciende automáticamente

Para la luminosidad:
Aclaración: la luminosidad la he interpretado como que si el día está despejado o soleado la luminosidad es máxima, en otro caso no lo es.

- Si la temperatura es menor que altaTemp(30º) pero mayor que ajaTemp (10º) y la luminosidad es máxima (es decir, Despejado o Soleado) se interpreta como que hace buen tiempo y se abrirán las ventanas automáticamente.
- Si la temperatura es mayor queMAXTemp(35º) y la luminosidad es máxima (es decir, Despejado o Soleado) se interpreta como que hace calor y se cerrarán las persianas automáticamente.

Para el viento:

- Si el viento es mayor al umbral maxViento y la ventana está abierta, está se cerrará

automáticamente.

Para las precipitaciones:

- Si el porcentaje de precipitaciones supera al umbral de maxprecipt(70) y la ventana está abierta se cerrará automáticamente.

Cuando se produce una alarma esta se emite al cliente para que la procese y la incluya en su interfaz.

Podemos ver un ejemplo de una función de gestión de alarmas:

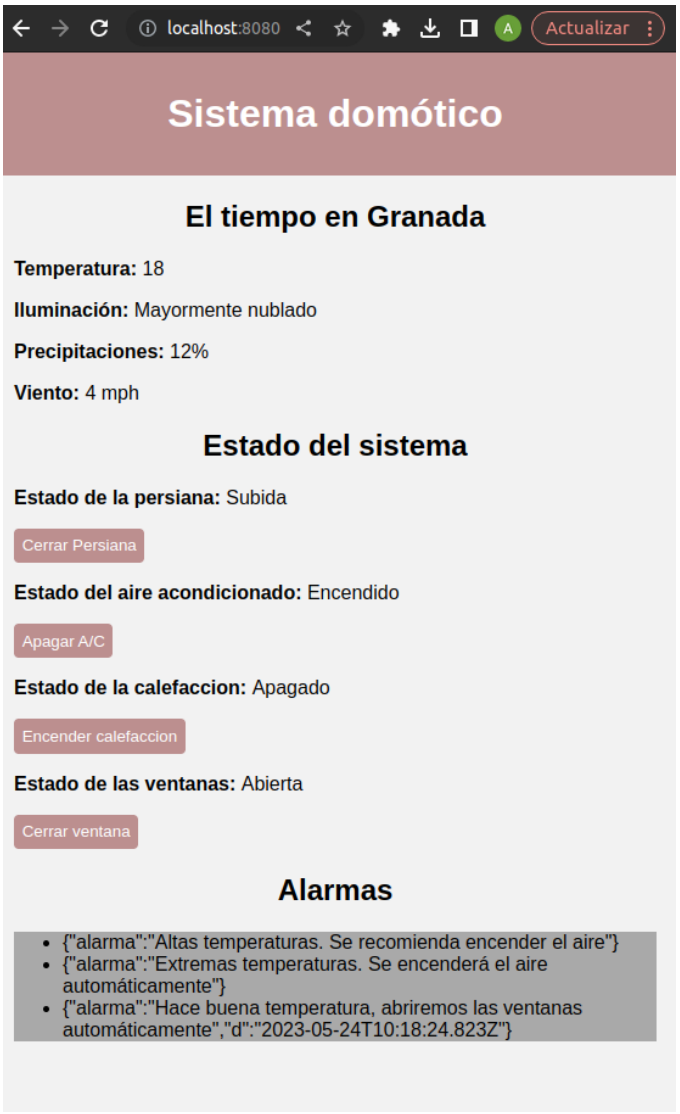
```
if(temp >= MAXTemp && AC=="Apagado"){
                alarma = "Extremas temperaturas. Se encenderá el aire automáticamente";
                collection.insertOne({alarma, d}, {safe:true}, function(err, result) {});
                io.sockets.emit('alarma', {alarma});
                AC = "Encendido";
                collection.insertOne({AC:"Encendido", d}, {safe:true}, function(err, result) {});
                io.sockets.emit('actualizarSistema', {AC:"Encendido", d});
              }
```

2.3. Cliente

Para el cliente he realizado una interfaz dónde se puede ver el estado de los sensores, de los activadores y la posibilidad de cambiar el estado de estos últimos.

Además le he añadido una hoja de estilo llamada estilo.css para que sea una interfaz más llamativa.

A continuación le muestro un ejemplo de cómo se vería:



La funcionalidad del cliente consiste en:

- Escuchar cuando hace click en un botón y cambiar sus valores, así como llamar al método add… correspondiente para actualizar el sistema.

```
var botonPersiana= document.getElementById("botonPersiana");
            var botonAC = document.getElementById("botonAC");
            var botonCalefaccion = document.getElementById("botonCalefaccion");
            var botonVentana = document.getElementById("botonVentana");

            botonPersiana.addEventListener("click", enviarPersiana);
            botonAC.addEventListener("click", enviarAC);
            botonCalefaccion.addEventListener("click", enviarCalefaccion);
            botonVentana.addEventListener("click", enviarVentana);

            function enviarPersiana(event) {
                event.preventDefault();
                var p = botonPersiana.innerText;
```

```
                var d = new Date();

                if(p == "Cerrar Persiana"){
                    botonPersiana.innerText = "Abrir Persiana";
                    socket.emit('addPersiana', {persiana: "Bajada", time: d});
                }else{
                    botonPersiana.innerText = "Cerrar Persiana";
                    socket.emit('addPersiana', {persiana: "Subida", time: d});
                }
            }
```

- Escucha el método alarma y actualiza para cuándo sea necesario modificarlo en el sistema.

```
 //Actualizamos las variables de estado y las añadimos a la interfaz de nuestro sistema
            function actualizar(data){
                if(data['temperatura']){
                    temp = data['temperatura'];
                    document.getElementById("temperatura").innerHTML = temp;
                };

                if(data['luminosidad']){
                    luz = data['luminosidad'];
                    document.getElementById("luminosidad").innerHTML = luz;
                };

                if(data['viento']){
                    viento = data['viento'];
                    document.getElementById("viento").innerHTML = viento;
                };

                if(data['precipitaciones']){
                    precipitaciones = data['precipitaciones'];
                    document.getElementById("precipitaciones").innerHTML =
precipitaciones;
                };

                if(data['persiana']){
                    persiana = data['persiana'];
                    document.getElementById("persiana").innerHTML = persiana;
                    if(persiana == "Bajada"){
                        botonPersiana.innerText = "Abrir Persiana";
                    }else{
                        botonPersiana.innerText = "Cerrar Persiana";
                    }
                }

                if(data['AC']){
                    AC = data['AC'];
                    document.getElementById("AC").innerHTML = AC;
                    if(AC == "Encendido"){
                        botonAC.innerText = "Apagar A/C";
                    }else{
                        botonAC.innerText = "Encender A/C";
                    }
                }
```

```
            if(data['calefaccion']){
                calefaccion = data['calefaccion'];
                document.getElementById("calefaccion").innerHTML = calefaccion;
                if(calefaccion == "Encendido"){
                    botonCalefaccion.innerText = "Apagar calefaccion";
                }else{
                    botonCalefaccion.innerText = "Encender calefaccion";
                }
            }

            if(data['ventana']){
                ventana = data['ventana'];
                document.getElementById("ventana").innerHTML = ventana;
                if(ventana == "Abierta"){
                    botonVentana.innerText = "Cerrar ventana";
                }else{
                    botonVentana.innerText = "Abrir ventana";
                }
            }
        }

        //Añadimos las alarmas
        function addAlarma(datos) {
            var item = document.createElement('li');
            item.innerHTML = JSON.stringify(datos);
            document.getElementById("listaAlarmas").appendChild(item);
            console.log("alarmaRecibida");
        }

        socket.on('alarma', function(data){
            addAlarma(data);
        });

        socket.on('actualizarSistema', function(data) {
            actualizar(data);
        })
```

- Llama a cargarDatos una primera vez cuando se inicia el sistema y luego lo llama cada dos minutos para comprobar que el valor de los sensores no se ha modificado.

```
//Cargamos los datos al iniciar el sistema
            socket.emit('cargarDatos');

 //Actualizamos los datos cada 2 min
            setInterval(socket.emit('cargarDatos'), 120000);
```

2.4. Ejecución

Para ejecutar el sistema domótico únicamente hay que ejecutar el comando:

```
node servidor.js
```

A continuación, abrir el navegador http://localhost:8080.