

## PRÁCTICA 4: Node.JS

### 1-Aclaraciones

Antes de empezar voy a explicar cosas que se hacen tanto en los ejemplos como en el ejercicio a realizar para no repetirme.

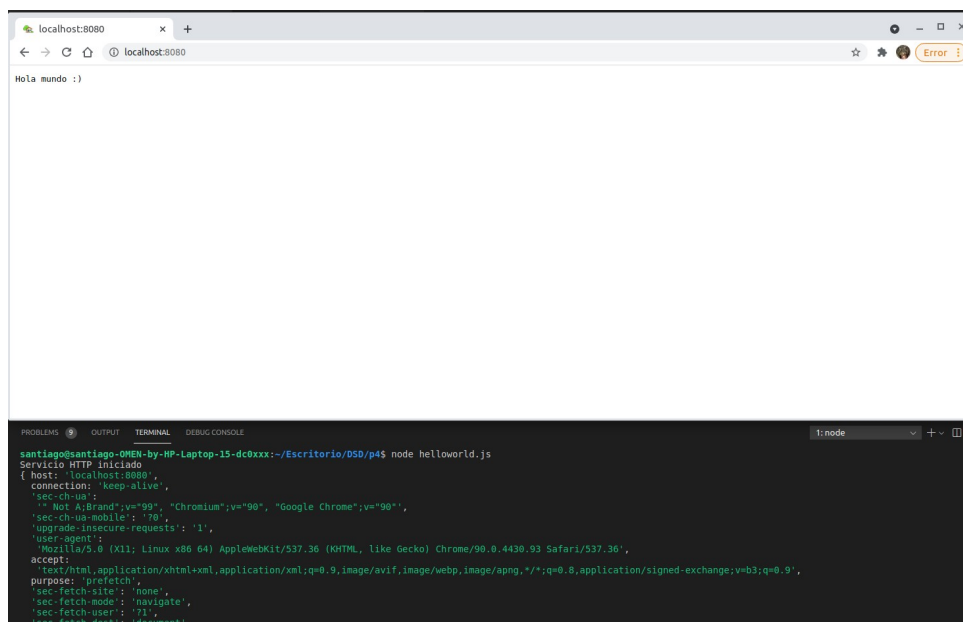
Para iniciar el servidor hay que irse primero a la localización del archivo donde esta el fichero js y ejecutar el comando **node fichero.js**, por ejemplo, para el servidor domático habría que poner en mi caso **cd /home/santiago/Escritorio/DSD/p4/Sistema domótica** y lanzar el servidor mediante **node servidor.js** . Por último, para acceder al contenido del servidor en cualquier buscador ponemos la siguiente url <http://localhost:8080/> , obviamente se tienen que tener instalado Node.js, Socket.io y mongodb.

Otra cosa a tener en cuenta, para la explicación del ejercicio a realizar sobre el servidor domótico se ha realizado de forma más resumida, ya que en el código se encuentran documentados la funcionalidad de cada parte del código de forma breve, por lo que no me extenderé mucho en algunas funciones de relevancia secundaria.

### 2-Ejemplos

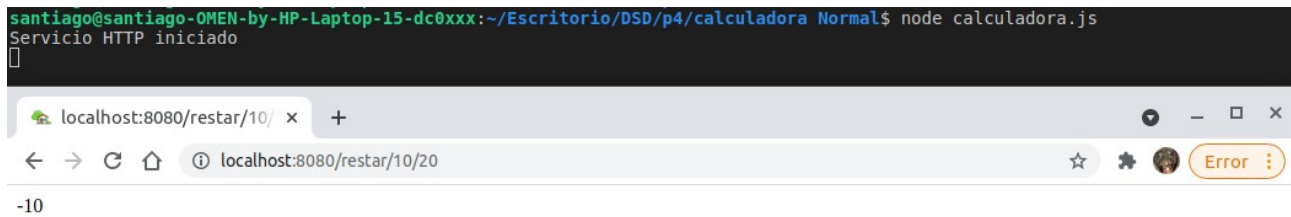
#### 2.1-Ejemplo 1

Es el más básico, es el mítico helloworld de cualquier inicio, consiste en crear un servidor escuchando en el puerto 8080 y con módulo http (como en el resto de ejemplos y ejercicio final) y escribir “Hola mundo” en la cabecera.



## 2.2-Ejemplo 2

En este ejemplo se simula una calculadora mediante peticiones REST del tipo “<http://localhost:8080/operacion/operando1/operando2>”.

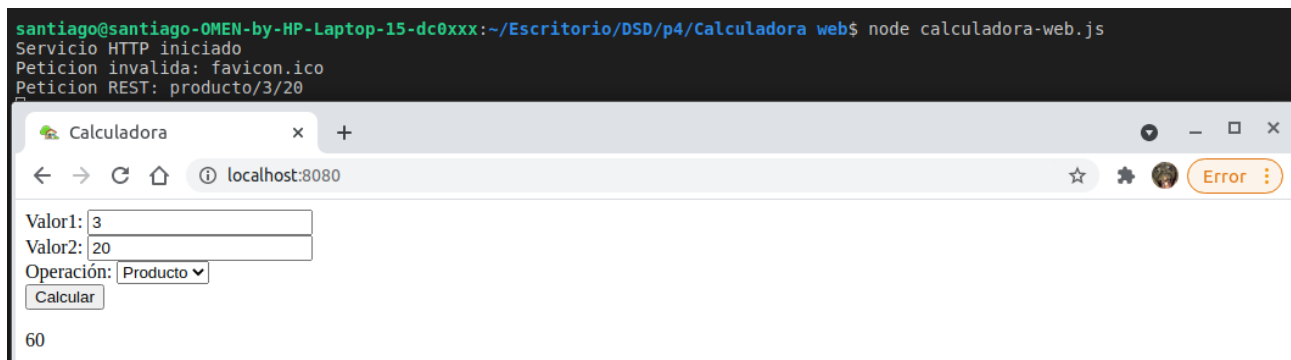


En esta imagen podemos observar que le decimos al servidor que nos reste 10 menos 20, para ello añadimos a la url de siempre `/restar/10/20`. El servidor divide esta parte de la url mediante **split** en variables (operación, valor 1 y valor 2) que son pasadas una función básica que interpreta estas variables y devuelve un resultado que se pone en la página, estas variables tienen formato HTML por lo que no hace falta declarar un fichero html para pasar estos valores.

## 2.3-Ejemplo 3

Este ejemplo ya utiliza dos ficheros, el servidor se sigue reciclando cosas como la función básica que calcula los valores del ejemplo anterior, la diferencia esta en la forma de obtener el tipo de operación y los operandos. Para ello el servidor utiliza el módulo `fs` que permite realizar operaciones de entrada/salida sobre el sistema de fichero en el servidor.

En el fichero html consta de un formulario donde se introducen los dos operandos y el tipo de operación y envía una petición REST (añadiendo la la url de tipo `/operando/valor1/valor2` como en el ejemplo anterior) al servidor para que le devuelva el resultado y lo pueda mostrar en pantalla.



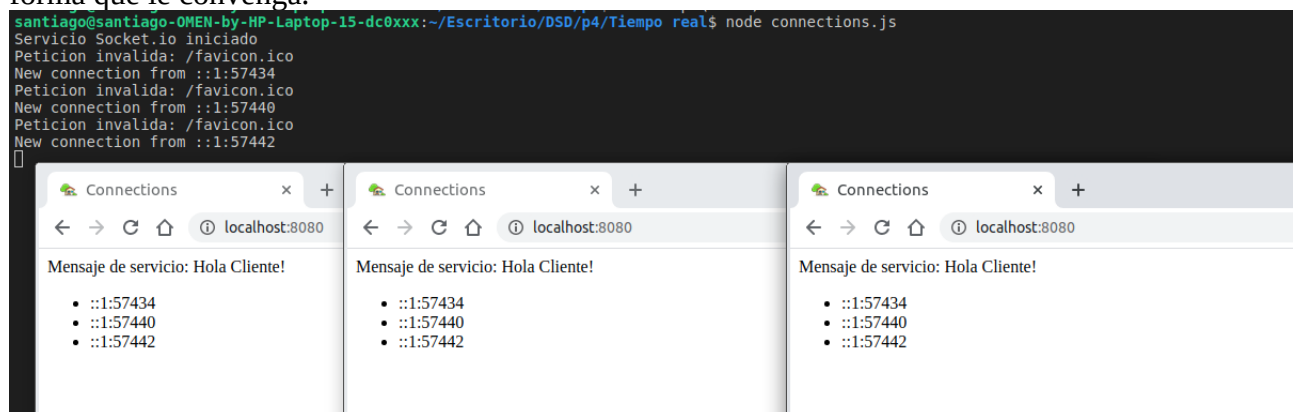
Lo de petición inválida que aparece no altera el funcionamiento del servidor y se soluciona añadiendo un icono con extensión `.ico` (para el ejercicio de la práctica si lo he utilizado).

## 2.4-Ejemplo 4

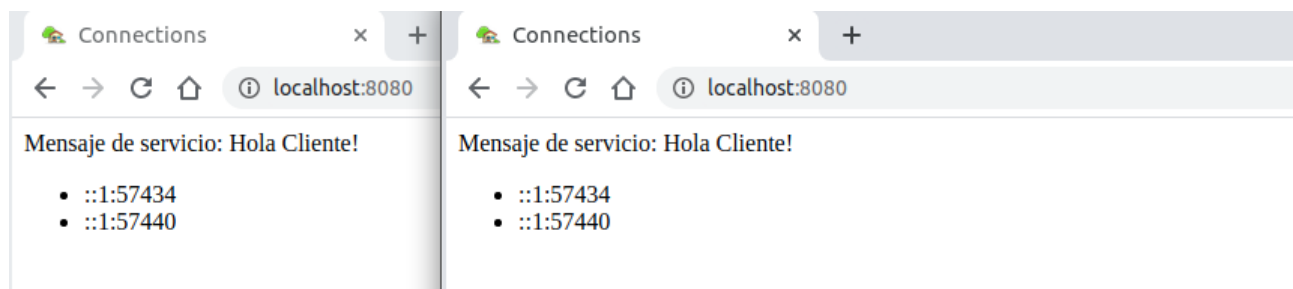
La primera diferencia de este ejemplo con los anteriores es que se utiliza `Socket.io`, que nos permite mediante sockets o nodos realizar conexiones permanentes entre un servidor y los clientes, pudiéndose comunicar entre ellos de la forma más sencilla.

Consta de dos ficheros, el servidor donde se crea el servidor igual que en los ejemplos anteriores, pero la implementación de la funcionalidad es distinta. Para enviar una petición a otro nodo se utiliza **nodo.emit('identificador de la petición',datos a enviar)** y luego habrá un nodo escuchando/esperando a esa petición mediante **nodo.on('identificador de la petición',function('datos a enviar'))**.

Para este ejemplo cada vez que un cliente accede al servidor, este último envía a todos los usuarios activos del sistema una lista de todos los cliente, el cliente recibe la lista y lo muestra en una lista en el html. Luego el servidor permanecerá escuchando las peticiones de los clientes como el output-evt que saluda al cliente y el disconnect que elimina al usuario de la lista de usuarios, ambos métodos acaban emitiendo al cliente el resultado de dicha operación, que este mostrara en su html de la forma que le convenga.



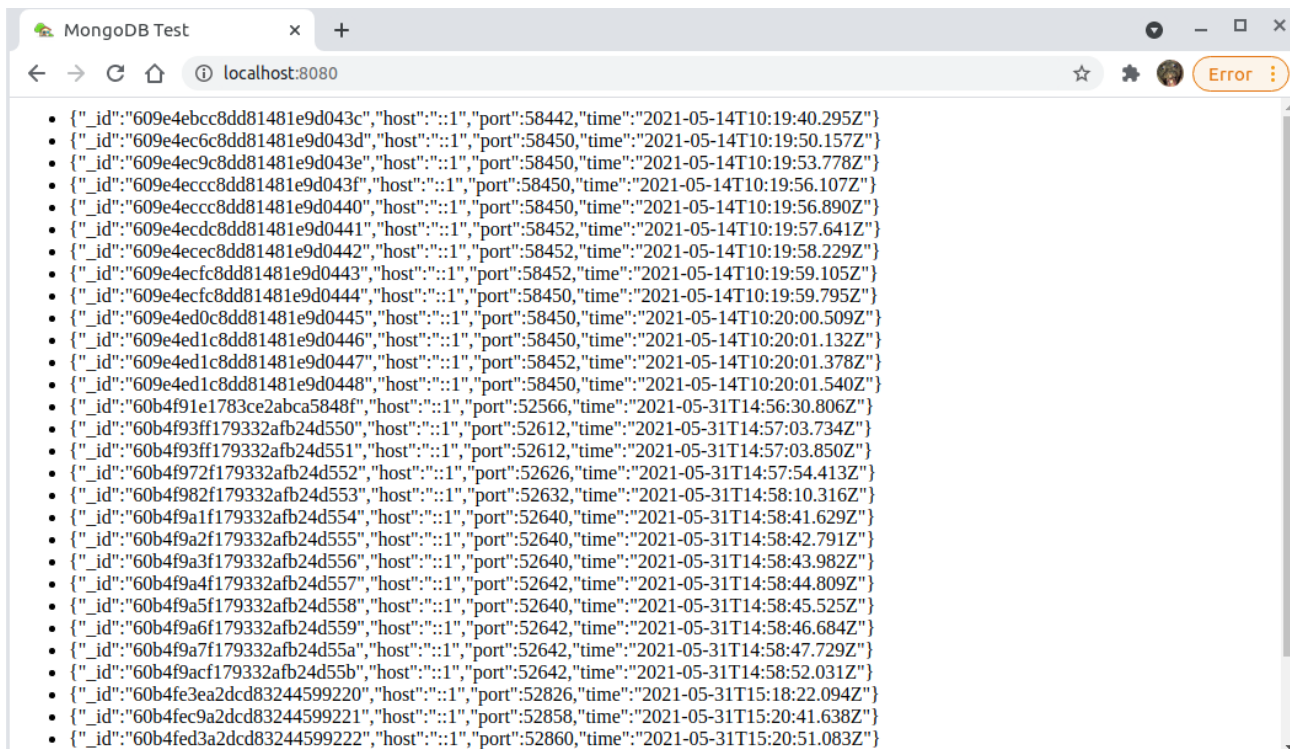
En esta imagen accedemos con tres usuarios distintos al servidor, y cada uno recibe su saludo y la lista de todos los usuarios si ahora cerramos uno, la lista se reducirá en uno para los otros dos cliente restantes.



## 2.5-Ejemplo 5

Este último ejemplo utiliza sockets como en el ejercicio anterior, pero esta vez utiliza también mongodb, una base de datos de tipo NoSQL, que guarda información no estructurada dándonos más albedrío a la hora de manejar datos sin necesidad de utilizar comando de SQL para crear tablas, actualizarlas, insertar datos etc. Este último ejemplo es el más importante ya que es el que más se parece al ejercicio del servidor doméstico.

El funcionamiento de los sockets es el mismo que en el ejemplo anterior pero esta vez no utilizamos un array de usuarios declarada en el servidor sino que lo obtenemos de mongodb, al ingresar un nuevo usuario introducimos sus datos en la base de datos creada mediante insert, para la lista de usuarios se obtiene mediante un find('dato a buscar'), resultado de estas consultas son enviadas al cliente y este las utilizara en un html como le convenga.



```
• { "_id": "609e4ebcc8dd81481e9d043c", "host": "::1", "port": 58442, "time": "2021-05-14T10:19:40.295Z" }
• { "_id": "609e4ec6c8dd81481e9d043d", "host": "::1", "port": 58450, "time": "2021-05-14T10:19:50.157Z" }
• { "_id": "609e4ec9c8dd81481e9d043e", "host": "::1", "port": 58450, "time": "2021-05-14T10:19:53.778Z" }
• { "_id": "609e4ecc8dd81481e9d043f", "host": "::1", "port": 58450, "time": "2021-05-14T10:19:56.107Z" }
• { "_id": "609e4eccc8dd81481e9d0440", "host": "::1", "port": 58450, "time": "2021-05-14T10:19:56.890Z" }
• { "_id": "609e4ecd8dd81481e9d0441", "host": "::1", "port": 58452, "time": "2021-05-14T10:19:57.641Z" }
• { "_id": "609e4ec8dd81481e9d0442", "host": "::1", "port": 58452, "time": "2021-05-14T10:19:58.229Z" }
• { "_id": "609e4ecfc8dd81481e9d0443", "host": "::1", "port": 58452, "time": "2021-05-14T10:19:59.105Z" }
• { "_id": "609e4ecfc8dd81481e9d0444", "host": "::1", "port": 58450, "time": "2021-05-14T10:19:59.795Z" }
• { "_id": "609e4ed0c8dd81481e9d0445", "host": "::1", "port": 58450, "time": "2021-05-14T10:20:00.509Z" }
• { "_id": "609e4ed1c8dd81481e9d0446", "host": "::1", "port": 58450, "time": "2021-05-14T10:20:01.132Z" }
• { "_id": "609e4ed1c8dd81481e9d0447", "host": "::1", "port": 58452, "time": "2021-05-14T10:20:01.378Z" }
• { "_id": "609e4ed1c8dd81481e9d0448", "host": "::1", "port": 58450, "time": "2021-05-14T10:20:01.540Z" }
• { "_id": "60b4f91e1783ce2abca5848f", "host": "::1", "port": 52566, "time": "2021-05-31T14:56:30.806Z" }
• { "_id": "60b4f93ff179332afb24d550", "host": "::1", "port": 52612, "time": "2021-05-31T14:57:03.734Z" }
• { "_id": "60b4f93ff179332afb24d551", "host": "::1", "port": 52612, "time": "2021-05-31T14:57:03.850Z" }
• { "_id": "60b4f972f179332afb24d552", "host": "::1", "port": 52626, "time": "2021-05-31T14:57:54.413Z" }
• { "_id": "60b4f982f179332afb24d553", "host": "::1", "port": 52632, "time": "2021-05-31T14:58:10.316Z" }
• { "_id": "60b4f9a1f179332afb24d554", "host": "::1", "port": 52640, "time": "2021-05-31T14:58:41.629Z" }
• { "_id": "60b4f9a2f179332afb24d555", "host": "::1", "port": 52640, "time": "2021-05-31T14:58:42.791Z" }
• { "_id": "60b4f9a3f179332afb24d556", "host": "::1", "port": 52640, "time": "2021-05-31T14:58:43.982Z" }
• { "_id": "60b4f9a4f179332afb24d557", "host": "::1", "port": 52642, "time": "2021-05-31T14:58:44.809Z" }
• { "_id": "60b4f9a5f179332afb24d558", "host": "::1", "port": 52640, "time": "2021-05-31T14:58:45.525Z" }
• { "_id": "60b4f9a6f179332afb24d559", "host": "::1", "port": 52642, "time": "2021-05-31T14:58:46.684Z" }
• { "_id": "60b4f9a7f179332afb24d55a", "host": "::1", "port": 52642, "time": "2021-05-31T14:58:47.729Z" }
• { "_id": "60b4f9ac179332afb24d55b", "host": "::1", "port": 52642, "time": "2021-05-31T14:58:52.031Z" }
• { "_id": "60b4fe3ea2dcd83244599220", "host": "::1", "port": 52826, "time": "2021-05-31T15:18:22.094Z" }
• { "_id": "60b4fec9a2dcd83244599221", "host": "::1", "port": 52858, "time": "2021-05-31T15:20:41.638Z" }
• { "_id": "60b4fed3a2dcd83244599222", "host": "::1", "port": 52860, "time": "2021-05-31T15:20:51.083Z" }
```

En esta imagen aparece toda la lista de usuarios que han estado en el sistema, al guardarse en una base de datos, estos usuarios no se llegan a perder aunque se apague el servidor, para volver a tener una lista vacía habría que borrar todas las instancias de la colección de mongodb.

### 3-Ejercicio servidor doméstico

Como me comentado al principio del documento todas las funciones de todos los ficheros se encuentran documentados, por lo que solo me centrare en explicar lo más importante, es decir, en la parte del servidor.

Antes de nada mi proyecto consta de tres ficheros:

- **Servidor.js:** donde esta implementada la base de datos y la estructura principal del ejercicio
- **sensores.html:** Formulario donde se pueden simular los valores de los sensores y ver la lista de los usuarios activos del sistema (<http://localhost:8080/>).
- **PaginaUsuario.html:** Página del usuario donde se verá el estado de los actuadores y sensores de la casa , historial de alarmas y el envío de correo (<http://localhost:8080/PaginaUsuario.html>)

#### 3.1-Implementación

El servidor utiliza los mismos recursos que en los ejemplos anteriores.

El servidor tiene una variables declaradas que guardan los valores de los sensores, los estados de los actuadores y umbrales del agente, todos ellos inicializados a un valor, por otro lado también se tiene un array de las alarmas del agente y de los usuarios activos en el sistema.

Los valores de los sensores son la temperatura, luminosidad, viento y humedad.

Tanto el viento como la humedad son valores obtenido utilizando la api de openWeather.

```
//Petición para obtener los datos de Granada en openWeather de los cuales nos quedamos con la humedad y el viento
request(urlApi, function(err, response, body){
  if(err){
    console.log('error:',err);
  }
  else{
    let info= JSON.parse(body);
    humedad=info.main.humidity;
    viento=info.wind.speed;
  }
});
```

Cuando se inicia el servidor este manda una petición a openWeather, si no hay fallo se obtendrá un data en JSON, el cual decodificamos y obtenemos el valor de la humedad y velocidad del viento.

El estado de los actuadores son las persianas(subir/bajar), el aire acondicionado(apagar/encender), los toldos(subir/bajar) , los aspersores(encender/apagar) y el agente(apagar/encender).

La creación del servidor es la misma que en los ejemplo anteriores.

Para este servidor se ha creado una colección en mongodb donde se guardaran los cambios en los sensores. Siempre que un usuario ingrese al sistema se ingresara su información en el array de usuarios y sera enviado al socket de sensores que al recibir la petición actualizara la lista de usuarios en el html, esta implementación es igual a la del ejemplo 4.

Nada más ingresar el usuario también se envían los valores a todos los usuarios para que estos nada más meterse tengan todos los valores actuales y no los inicializados anteriormente. A partir de aquí el servidor escuchara peticiones que recibirá de los sensores para cuando se cambie un valor y del usuario para cuando se cambie el estado de los actuadores u otra función.

Si se realiza un cambio en los actuadores, el servidor actualiza la variable y llama al método **enviarDatos()**, este método es el mas importante ya que hace un emit a todos los usuarios activos del sistema sobre los nuevos cambios, de esta forma todos los usuarios constaran del cambio al momento aunque no lo hayan realizado.

```
//Acción donde se recibe y guarda la actuacion sobre las persianas

client.on('accion-persiana', function (data) {
  estadoPersianas = data.accion;
  enviarDatos();
});

//Acción donde se recibe y guarda la actuacion sobre el aire
client.on('accion-aire', function (data) {
  estadoAire = data.accion;
  enviarDatos();
});

//Acción donde se recibe y guarda la actuacion sobre el agente
client.on('accion-agente', function (data) {
  estadoAgente = data.accion;
  enviarDatos();
});

//Acción donde se recibe y guarda la actuacion sobre los aspersores
client.on('accion-aspersor', function (data) {
  estadoAspersor = data.accion;
  enviarDatos();
});
```



Si se realiza un cambio en los sensores, el servidor insertara en la base datos el sensor que ha recibido cambio, el nuevo valor y la fecha de cuando se ha cambiado. Luego se actualizara las variables del servidor y se llamara al agente el cual es un método que compara los valores actuales con los umbrales (si esta apagado por el usuario no lo hará) y introducirá mensajes de alerta en el vector de alarmas que luego recibirá el usuario para mostrarlas como lista en el html. A continuación las condiciones que tiene en cuenta el agente.

```
if (estadoAgente=="encender") {  
  if (temperatura >= umbralTemperatura && luminosidad >= umbralMaxLuminosidad && estadoPersianas == "subir") {  
    alarmas.push("La temperatura y luminosidad superan el umbral máximo por lo que se han cerrado las persianas");  
    estadoPersianas = "bajar";  
  }  
  if (temperatura >= umbralTemperatura && estadoPersianas == "subir") {  
    alarmas.push("La temperatura supera el umbral máximo por lo que se han cerrado las persianas");  
    estadoPersianas = "bajar";  
  }  
  if (temperatura >= 40 && estadoAire == "apagar") {  
    alarmas.push("La temperatura supera el umbral máximo por lo que se ha encendido el aire");  
    estadoAire = "encender";  
  }  
  if(temperatura <=15 && estadoAire=="encender")  
  {  
    alarmas.push("La temperatura es baja por lo que se ha desactivado el aire");  
    estadoAire = "apagar";  
  }  
  if (luminosidad >= umbralMaxLuminosidad && estadoPersianas == "subir") {  
    alarmas.push("La luminosidad supera el umbral máximo por lo que se han cerrado las persianas");  
    estadoPersianas = "bajar";  
  }  
  if(estadoAspersor=="apagar" && humedad<=umbralHumedad)  
  {  
    alarmas.push("La humedad del jardín es muy baja por lo que se han activado los aspersores");  
    estadoAspersor="encender";  
  }  
  if(estadoAspersor=="encender" && humedad>=70)  
  {  
    alarmas.push("La humedad del jardín es alta por lo que se han desactivado los aspersores");  
    estadoAspersor="apagar";  
  }  
  if(estadoToldo=="subir" && viento>=35) {  
    alarmas.push("La velocidad del viento es alta por lo que se han bajado los toldos");  
    estadoToldo="bajar";  
  }  
  if(estadoToldo=="bajar" && viento<30 && temperatura>=35)  
  {  
    alarmas.push("La velocidad del viento es baja y la temperatura es alta por lo que se han subido los toldos");  
    estadoToldo="subir";  
  }  
}
```

Otras funcionalidades extra del servidor es que el cliente puede eliminar la lista de alarmas, por si le molesta, y la posibilidad de enviar un correo del estado de la casa a un correo, para esto se ha tenido que definir un transportador con una cuenta de email nueva por casos de seguridad, esta sera la que enviara los correos.

```
//Declaramos el transportador para enviar correos, con una cuenta creada unicamente para enviar mensaje a los usuarios  
const transporter = nodemailer.createTransport({  
  service: 'gmail',  
  auth: {  
    user: 'servidordomativodsd3@gmail.com',  
    pass: 'claveDSD'  
  }  
});
```

Luego cuando el servidor reciba una petición de enviar correo, creara un mensaje con los datos de la casa, define el destinatario con el correo que recibe del usuario y enviá el correo, en caso de éxito el servidor notifica al usuario con un emit y este mostrara al usuario un mensaje de correo enviado con éxito.

```

client.on('enviar-correo',function(data){

    //Se crea el mensaje a enviar

    let mensaje="El estado de la casa es: \nTemperatura--> "+temperatura+"\nLuminosidad-->"+luminosidad+"\nHumedad-->"+humedad+
    "\nViento-->"+viento+"\nPersianas-->"+estadoPersianas+"\nAire-->"+estadoAire+"\nAspersores-->"+estadoAspersor+"\nToldos-->"+estadoToldo+"\nAgente-->"+estadoAgente;

    //Se define las opciones del email(escriptor, destino, asunto y el mensaje)
    const mailOptions={
        from:'servidordomativods3@gmail.com' ,
        to: data.correo,
        subject:'Informe estado de la casa domótica',
        text: mensaje
    };

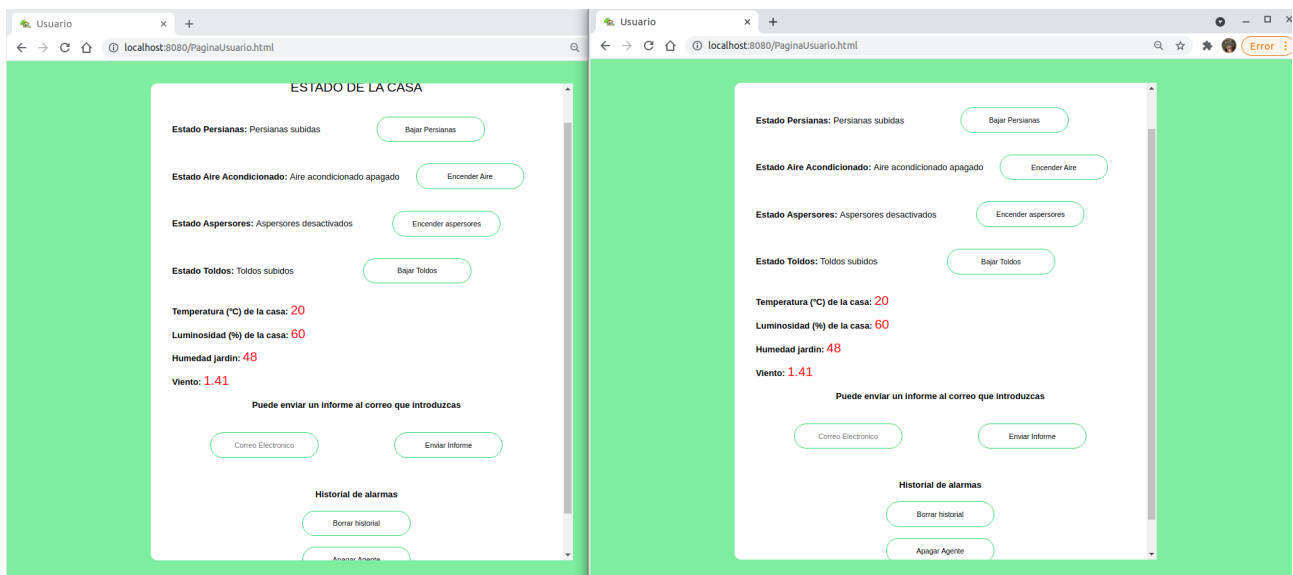
    //Se manda el email, en caso de acierto se notifica al usuario que ha realizado la petición
    transporter.sendMail(mailOptions,function(error, info){
        if(error)
        {
            console.log(error);
        }else{
            client.emit('confirmar-correo',{});
        }
    });
});

```

En cuanto a la implementación de los htmls de los usuarios y sensores, no es necesario entrar en detalles, básicamente cada uno tiene sus formularios donde se realizaran acciones que el servidor recibirá y este enviara los cambios de nuevo a los usuarios, estos datos que recibe el usuario los muestra en el html de la mejor forma, en mi caso he puesto una buena retroalimentación y css medio bonito, por ejemplo si pulsa el botón de bajar persianas cambia todos los mensajes relacionados con este como “persianas bajadas”, “subir persianas” en el botón etc.

## 3.2-Capturas

Para las capturas con el ejemplo de ejecución, usare tres usuarios, uno que utiliza los sensores y otros dos que utilizan la página de usuario.



Ambos usuarios al meterse de primeras.

## SENSORES

Temperatura

Enviar Temperatura

Luminosidad

Enviar Luminosidad

Humedad

Enviar Humedad

Viento

Enviar Viento

### Usuarios utilizando el sistema

- Usuario con host: ::1 y puerto: 57654 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57710 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57710 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57720 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57728 ha accedido al sistema el día 6 del 5 a las 23:42



## SENSORES

40

Enviar Temperatura

Luminosidad

Enviar Luminosidad

Humedad

Enviar Humedad

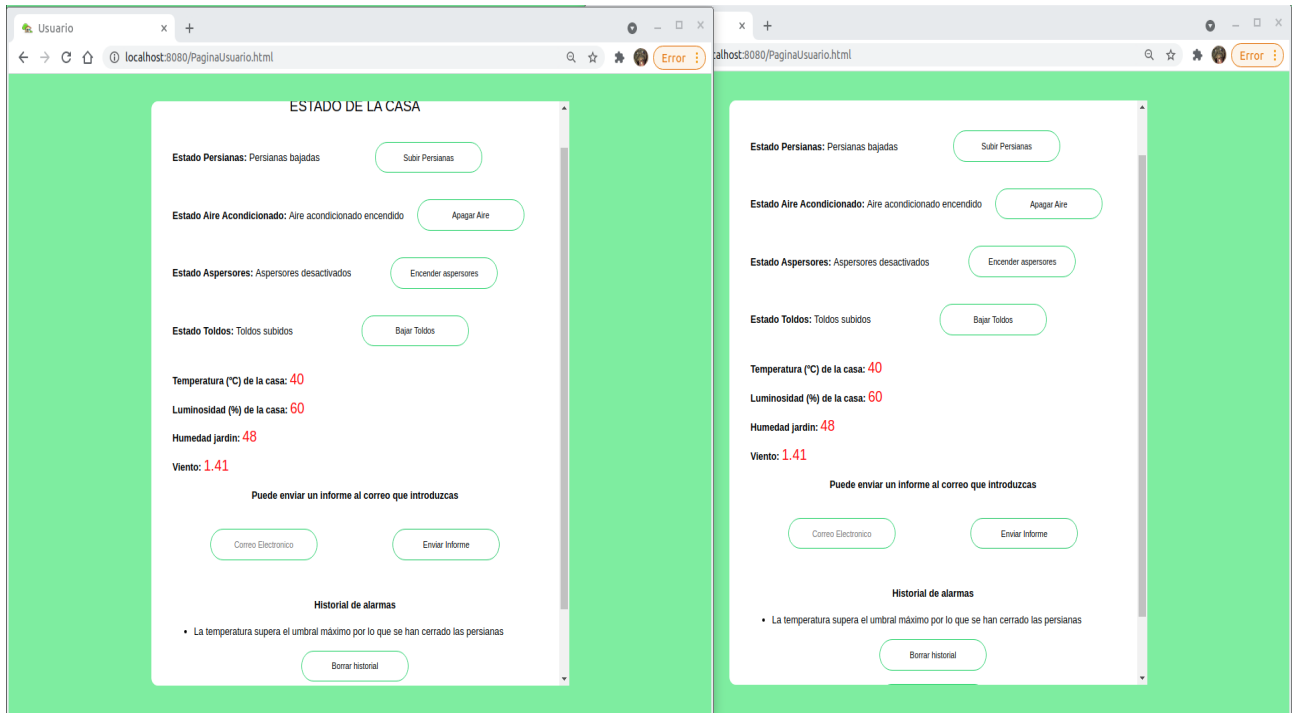
Viento

Enviar Viento

### Usuarios utilizando el sistema

- Usuario con host: ::1 y puerto: 57654 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57710 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57710 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57720 ha accedido al sistema el día 6 del 5 a las 23:42
- Usuario con host: ::1 y puerto: 57728 ha accedido al sistema el día 6 del 5 a las 23:42

Probamos una funcionalidad del agente cambiando la temperatura a 40.



Apenas se aprecia pero ambos usuarios reciben las nueva temperatura les salta una alarma que cierra las persianas y están se bajan.

Foto con más detalle:

ESTADO DE LA CASA

Estado Persianas: Persianas bajadas

Subir Persianas

Estado Aire Acondicionado: Aire acondicionado encendido

Apagar Aire

Estado Aspersores: Aspersores desactivados

Encender aspersores

Estado Toldos: Toldos subidos

Bajar Toldos

Temperatura (°C) de la casa: 40

Luminosidad (%) de la casa: 60

Humedad jardin: 48

Viento: 1.41

Puede enviar un informe al correo que introduzcas

Correo Electronico

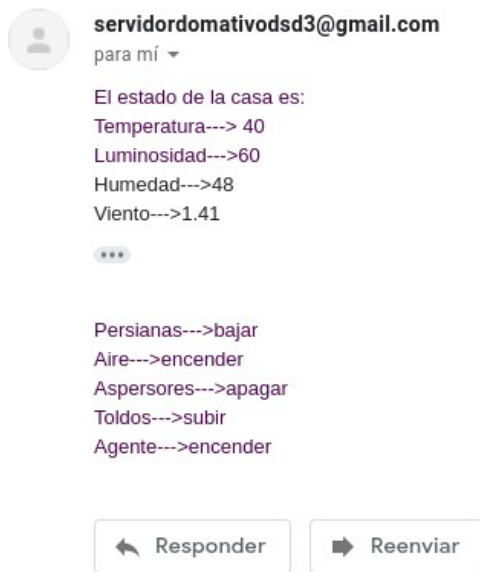
Enviar Informe

Historial de alarmas

- La temperatura supera el umbral máximo por lo que se han cerrado las persianas

Borrar historial

Si solicitamos un correo para el email [santiyagito22@gmail.com](mailto:santiyagito22@gmail.com), este correo recibirá el mensaje así:



El usuario que ha enviado el correo será notificado:

Temperatura (°C) de la casa: 40

Luminosidad (%) de la casa: 60

Humedad jardin: 48

Viento: 1.41

Puede enviar un informe al correo que introduzcas

santiyagito22@gmail.com

Enviar Informe

El correo ha sido enviado con éxito

Historial de alarmas

En cambio y lógicamente el otro usuario no le aparecerá la confirmación:

Estado Persianas: Persianas bajadas

Subir Persianas

Estado Aire Acondicionado: Aire acondicionado encendido

Apagar Aire

Estado Aspersores: Aspersores desactivados

Encender aspersores

Estado Toldos: Toldos subidos

Bajar Toldos

Temperatura (°C) de la casa: 40

Luminosidad (%) de la casa: 60

Humedad jardin: 48

Viento: 1.41

Puede enviar un informe al correo que introduzcas

Correo Electronico

Enviar Informe

Historial de alarmas

Es difícil mostrar todas las funcionalidades desde aquí, pongo una demostración básica, la más compleja la mostrare en la defensa.