

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

31-5-2022

Práctica 4: Node.JS

Uso de Node.js y MongoDB, y su
interconexión a través de Socket.io

Several thin, curved lines in dark blue and light grey that originate from the bottom left and sweep upwards and to the right.

Alberto Llamas González
3º GII, IS

1. Implementación de los ejemplos

Ejemplo 1: Hola mundo

El primer ejemplo que encontramos es el más sencillo, que nos enseña a utilizar Node.js. Como podemos observar en el código, se crea un servidor dónde se dice que imprima por pantalla "Hello World" y que se escuche en el puerto 8080. Si ejecutamos "`> nodejs helloworld.js`" y abrimos en un navegador una pestaña con la dirección <https://localhost:8080/> vemos que nos aparece por pantalla el mensaje 'Hola mundo'.

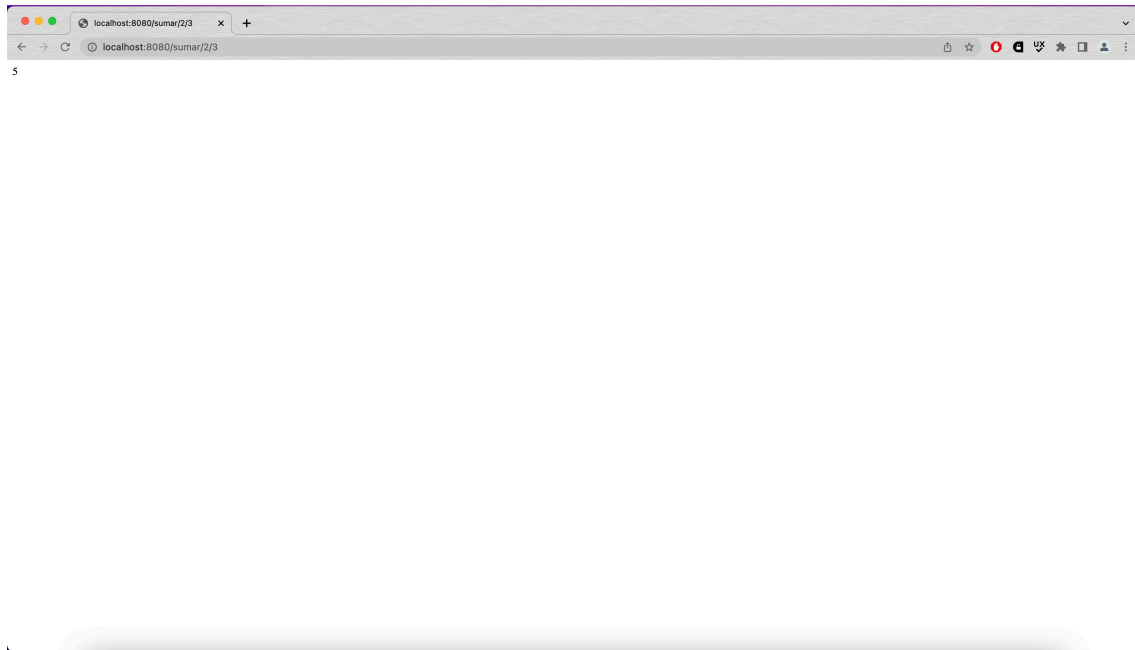


Ejemplo 2: Calculadora con interfaz tipo REST

En dicho ejemplo, se ejecutaba una calculadora que se define de la siguiente forma; <https://localhost:8080/sumar/2/3>. /sumar/2/3 son los parámetros que trata la dirección. Para extraer dichos parámetros, se divide en cadenas según el carácter / y dependiendo del primer fragmento, comparándolo con las palabras clave sumar, restar, producto y dividir se realiza la operación pertinente. Los otros dos parámetros son los operandos de la operación. Ejecutamos con:

```
node calculadora.js
```

Veamos un ejemplo:



Mejora de la calculadora

En esta ocasión, se mejora el servicio anterior proporciona una web con una interfaz para la calculadora en caso de que el usuario acceda desde el navegador a la dirección <http://localhost:8080/>. En esta ocasión tenemos la dirección anterior donde se sirve el archivo HTML (calc.html) asociado al archivo Node de la calculadora.

```
node calculadora-web.js
```



Ejemplo 3: aplicaciones en tiempo real con Socket.io

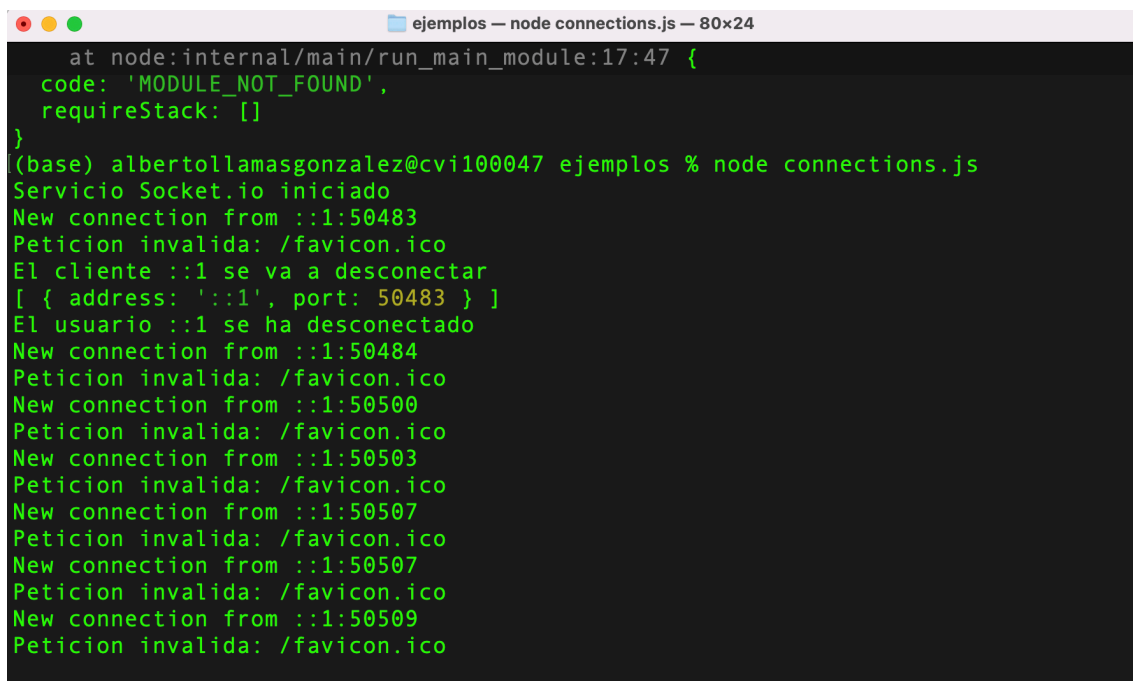
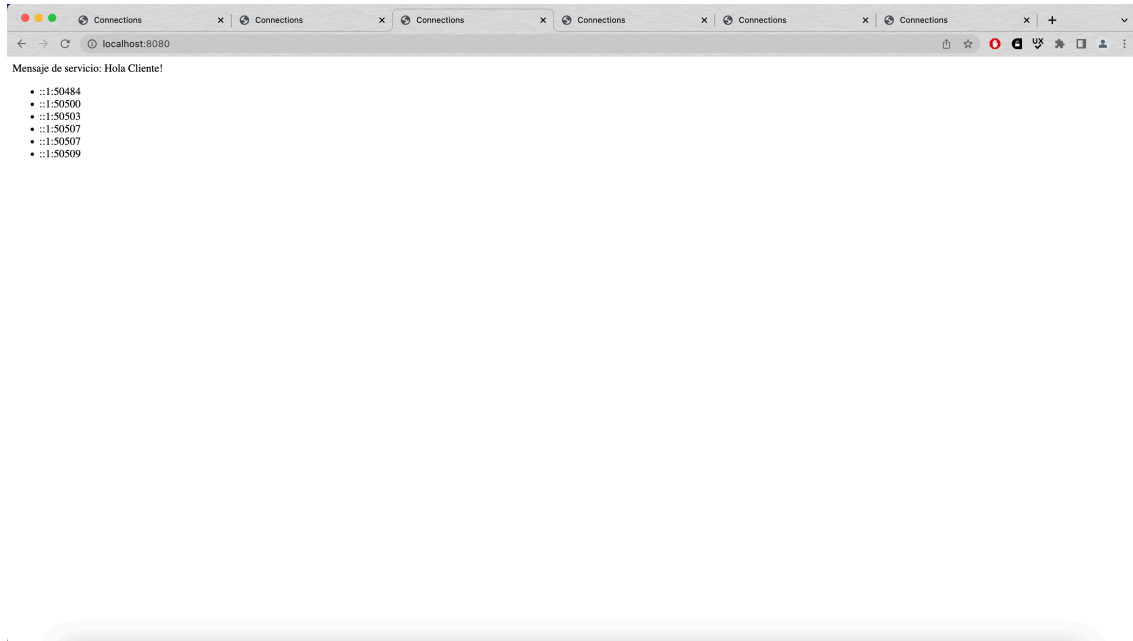
Recordemos, que para poder ejecutar el siguiente ejemplo es necesario instalar el paquete socket.io:

```
npm install socket.io
```

Para asegurarnos del funcionamiento tanto de los ejemplos como del ejercicio, debemos comprobar que en la carpeta donde estamos ejecutando el proyecto tenemos la carpeta **node_modules**, resultante del comando anterior. En el programa, podemos detectar mensajes que reciben y envían los clientes entre ellos, que además reciben mensajes globales. Podemos observar también que **io.sockets**, hace un broadcast a todos quienes reciben mensajes por ese socket. A partir de estos mensajes, se utiliza una página HTML que permite ver el estado del servidor (**connect/disconnect**). Además, se da un socket para ir añadiendo las direcciones en esa página de forma que, si entramos desde otra pestaña a la misma página (<https://localhost:8080>), dinámicamente

nos indica su dirección. En el servidor, se muestra una lista con las conexiones, desconexiones y una lista de conexiones.

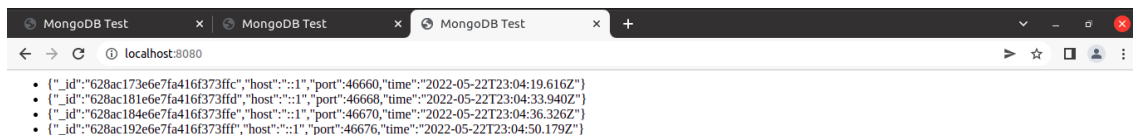
node connections.js



Ejemplo 4: uso de MongoDB desde Node.js

Este ejemplo, es una continuación del ejemplo de conexiones anterior pero ahora utilizamos MongoDB que es una base de datos de tipo NoSQL que significa que podemos guardar información de forma no estructurada.

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



En este ejemplo, tuve problemas ya que lo ejecutaba la primera vez sin ningún fallo, sin embargo, la segunda vez que lo ejecutaba, me daba error. Tras investigar con un compañero, nos dimos cuenta que era necesario eliminar la tabla creada en la base de datos de MongoDB para poder ejecutar de nuevo el ejemplo. Otra opción era cambiar el nombre de la base de datos creada cada vez, pero pensamos que es más sofisticado eliminar dicha base de datos. Para ello, ejecutamos lo siguiente:

```
mongo
// Una vez dentro del servicio MongoDB
show dbs // Para ver las bases de datos
use pruebaBaseDatos // Para usar dicha base de datos
db.dropDatabase(); // Para eliminar la base de datos
exit // Para salir de la base de datos
```

Y ya podríamos ejecutar de nuevo el ejemplo.

2. Ejercicio: Sistema Domótico

El objetivo de este ejercicio es implementar un sistema domótico compuesto por dos sensores (luminosidad y temperatura), dos actuadores (persiana y sistema de aire acondicionado) y un servidor que sirve para mostrar el estado y actuar sobre los elementos de la vivienda contando este además con un agente que notifica alarmas y realiza operaciones básicas.

2.1 Estructura de la carpeta del ejercicio

Nombre	Tamaño	Clase
agente.html	2 KB	Texto HTML
> node_modules	--	Carpeta
sensores.html	2 KB	Texto HTML
servidor.js	7 KB	JavaScr...t source
usuario.html	5 KB	Texto HTML

Es importante recalcar que para ejecutar el ejercicio, por si acaso, es necesario incluir dentro de la carpeta del ejercicio, **node_modules**.

2.2 Ejecución

Para ejecutar nuestro Sistema Domótico, lo realizamos igual que en los otros ejercicios:

```
node servidor.js
```

Posteriormente, consultamos <http://localhost:8080/sensores.html> para poder recoger los valores de los sensores. A continuación, accedemos a <http://localhost:8080/agente.html>, es decir, al agente, que nos mostrará, en caso de que todo esté funcionando bien, **activo**. Finalmente, accedemos a <http://localhost:8080/> para acceder a la información del Sistema Domótico. Hablemos ahora en detalle del funcionamiento. La explicación de los métodos se encuentra en el propio código.

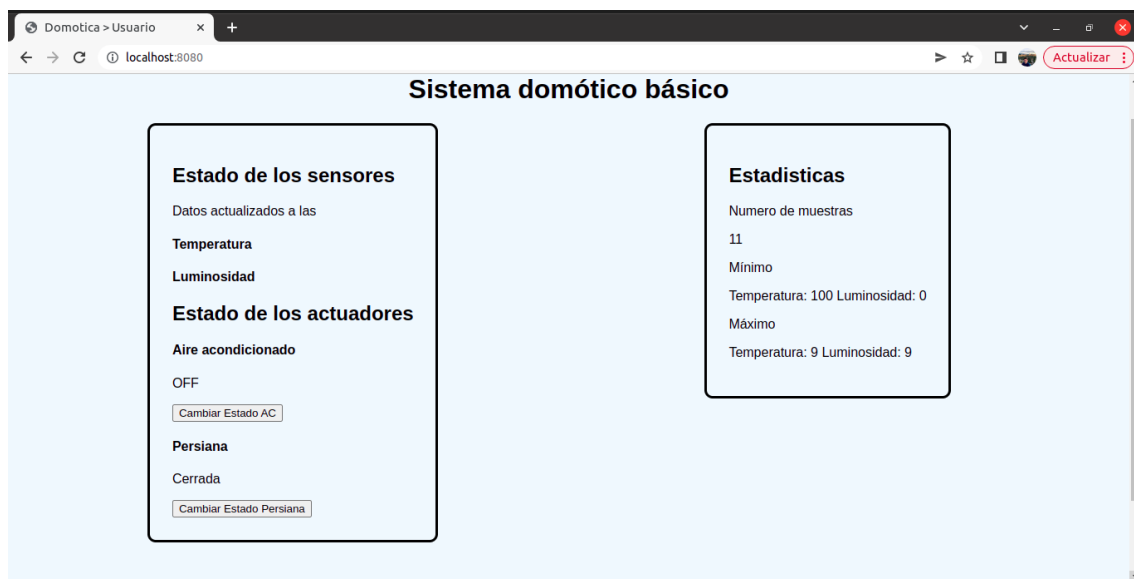
2.3 Servidor

En el servidor, realizamos las conexiones con Socket.io y MongoDB. El servidor, es el único que tiene acceso a la Base de Datos y donde se almacenan los estados de los actuadores, los cuales se van a modificar desde el servidor, aunque a través de los sockets, veremos que el agente va a solicitar la modificación.

Para realizar este ejercicio, se ha utilizado como base el código de los ejemplos, sobre todo los ejemplos 3 y 4.

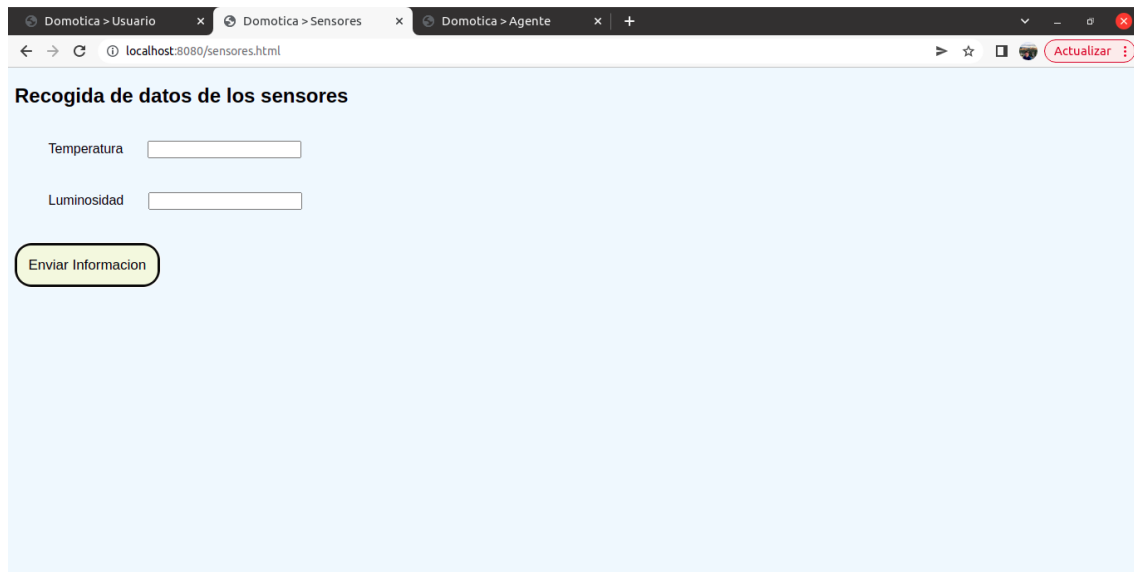
2.4 Usuarios

En la vista de los usuarios podemos ver los sensores junto con la marca de tiempo además de los estados de los actuadores (persiana y aire acondicionado) que se pueden modificar con los botones correspondientes. Al pulsar los botones, se llama desde ahí a los sockets correspondientes para cambiar sus estados.



2.5 Sensores

En el formulario donde se recogen los datos de los sensores, se introduce la temperatura y luminosidad y se pulsa en el botón **Enviar Información**. Al pulsar el botón, se llama a la función **enviarInformacion()** donde se cogen los valores de los campos y se envían a través de una llamada al socket para que el servidor actualice los valores en la vista del usuario.

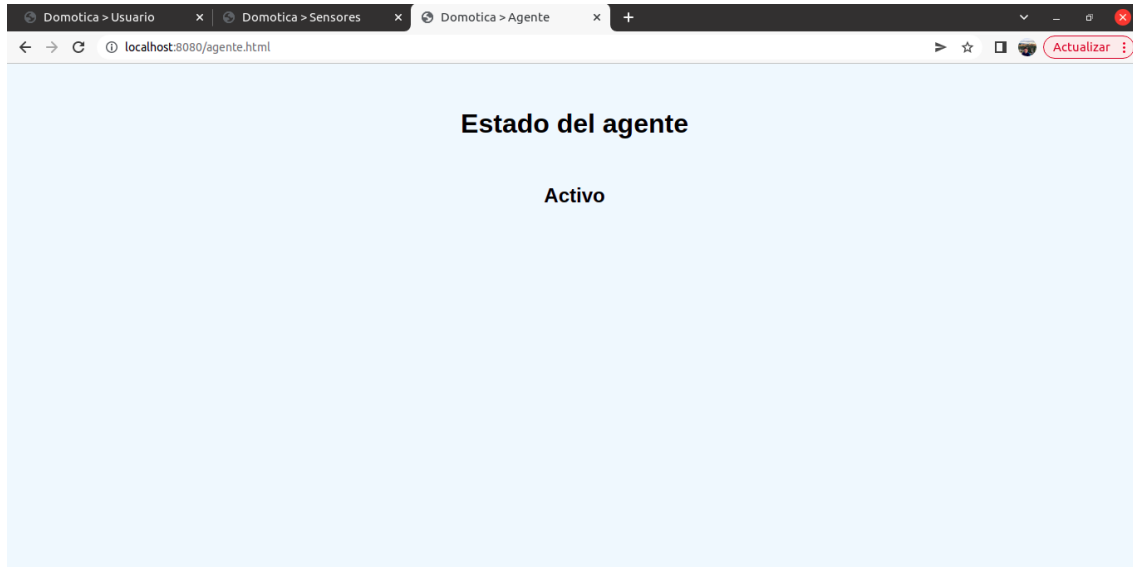


The screenshot shows a web browser with three tabs: 'Domotica > Usuario', 'Domotica > Sensores', and 'Domotica > Agente'. The address bar shows 'localhost:8080/sensores.html'. The page title is 'Recogida de datos de los sensores'. The form contains two input fields: 'Temperatura' and 'Luminosidad'. Below these fields is a button labeled 'Enviar Informacion'. In the top right corner of the browser window, there is a red button labeled 'Actualizar'.

2.6 Agente

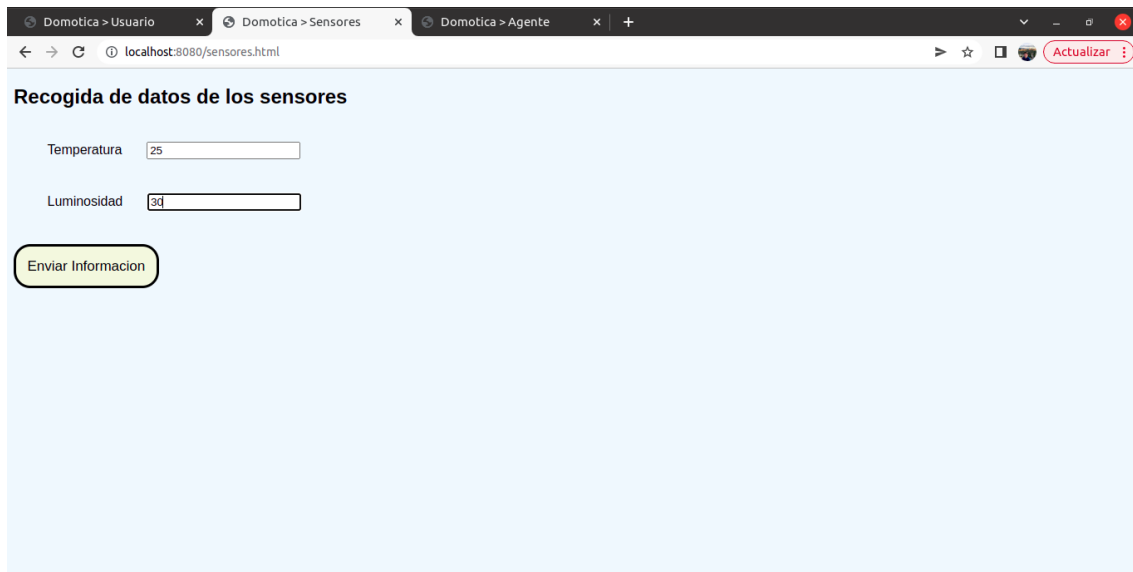
El agente, se encarga de emitir las alarmas cuando se supere el umbral máximo o mínimo y de observar los valores de la temperatura. Se realizarán las siguientes acciones si se superan esos límites:

- Si la temperatura es muy baja, se activa el aire acondicionado para calentar el hogar.
- Si hay mucha luz (luminosidad) se bajan las persianas y si hay poca luz, se suben las persianas.
- Si la temperatura es muy alta, se supone que hay mucho sol, por lo que se bajarán las persianas y se activará el aire acondicionado para enfriar el hogar.



Ejemplo de funcionamiento

Recogida de datos



Muestra de información

The screenshot shows a web browser with three tabs: 'Domotica > Usuario', 'Domotica > Sensores', and 'Domotica > Agente'. The address bar shows 'localhost:8080'. The page content is titled 'Sistema domótico básico' and is divided into two main sections:

- Estado de los sensores**:
 - Datos actualizados a las: Sat May 28 2022 11:01:53 GMT+0200 (hora de verano de Europa central)
 - Temperatura: 25
 - Luminosidad: 30
- Estado de los actuadores**:
 - Aire acondicionado: OFF, with a button 'Cambiar Estado AC'.
 - Persiana: Cerrada, with a button 'Cambiar Estado Persiana'.
- Estadísticas**:
 - Numero de muestras: 23
 - Mínimo: Temperatura: Luminosidad: 9
 - Máximo: Temperatura: 9 Luminosidad: 9

An 'Actualizar' button is visible in the top right corner of the page.

```
albertollamas@cvi040104: ~/Escritorio/DSD/Practica 4/practi...
[
  { address: '::1', port: 38694 },
  { address: '::1', port: 38714 }
]
El usuario ::1 se ha desconectado
New connection from ::1:38734
Petición invalida: /favicon.ico
El cliente ::1 se va a desconectar
[
  { address: '::1', port: 38694 },
  { address: '::1', port: 38714 },
  { address: '::1', port: 38734 }
]
El usuario ::1 se ha desconectado
Petición invalida: /favicon.ico
New connection from ::1:38736
Aire false
{
  temperatura: '25',
  luminosidad: '30',
  timeStamp: 'Sat May 28 2022 11:01:53 GMT+0200 (hora de verano de Europa centra
1)'
}
Promise { <pending> }
```

Recogida de datos con poca luminosidad y temperatura

Domotica > Usuario x Domotica > Sensores x Domotica > Agente x +

localhost:8080/sensores.html

Recogida de datos de los sensores

Temperatura

Luminosidad

Alarma correspondiente

Domotica > Usuario x Domotica > Sensores x Domotica > Agente x +

localhost:8080

Sistema domótico básico

¡Alerta! Temperatura alta ¡Alerta! Poca luz

Estado de los sensores

Datos actualizados a las
Sat May 28 2022 11:04:40 GMT+0200 (hora de verano de Europa central)

Temperatura
100

Luminosidad
10

Estado de los actuadores

Aire acondicionado
OFF

Persiana
Cerrada

Extra: Estadísticas

Como extra, se intentó realizar una muestra de estadísticas, pero no se consiguió que funcionase del todo bien, ya que muestra el total de muestras enviadas por los sensores, pero la mínima y máxima temperatura y luminosidad no la recoge correctamente y no se ha conseguido arreglar.

Extra 2: Otra aplicación

Como quería hacer algo distinto de extra, ya que creo que Node.js y Socket.io nos ofrecen muchas posibilidades de aplicaciones buenas concurrentes he decidido hacer un simple chat en tiempo real. Tenemos dos archivos, servidor.js y usuario.html muy similar al ejercicio anterior ya que he vuelto a tomar los ejemplos como base.

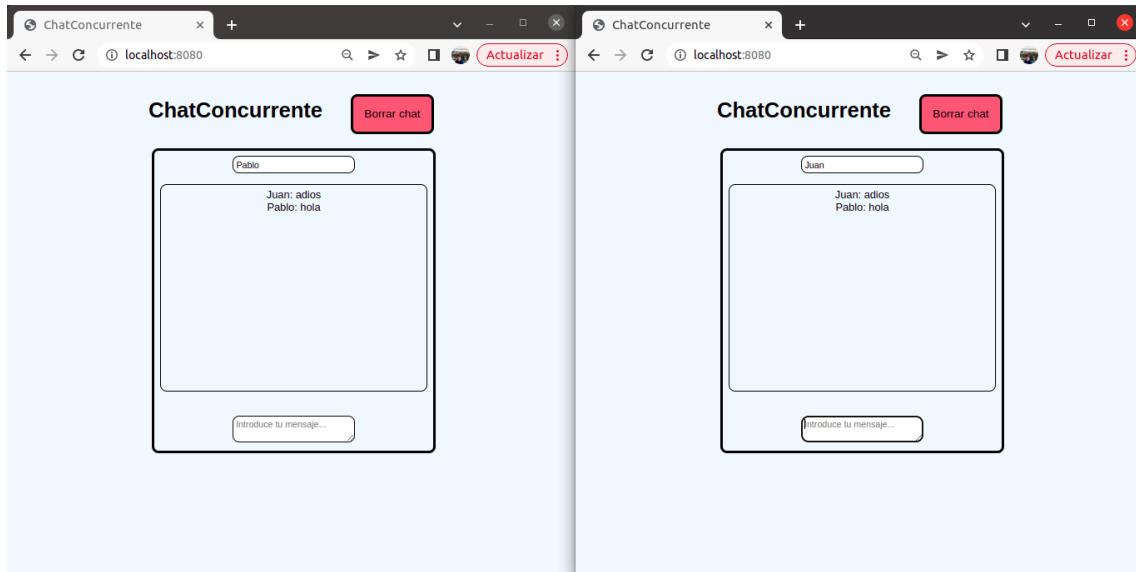
En el servidor se comprueban los mensajes y se insertan en la base de datos de MongoDB y se borran los mensajes del chat cuando se recibe del socket del usuario. En el usuario, nos conectamos a Socket.io y manejamos el envío de un mensaje, añadiéndolo al chat y manejamos la entrada de datos con los eventos de teclado. Además al pulsar el botón se emite por el socket que borre los mensajes del chat.

Ejecución y capturas del resultado

La aplicación se ejecuta de la misma forma que el ejercicio:

```
node servidor.js
```

Envío de mensajes



Borrado del chat

