UNIVERSIDAD
DE GRANADA

PRÁCTICA 4: NODE.JS

MEMORIA DE LA PRÁCTICA 4 DE DESARROLLO DE SISTEMAS DISTRIBUIDOS

Desarrollo de servicios web con Node.js, Socket.io y MongoDB

AUTOR: ÁLVARO LÓPEZ VERGARA

22 Mayo 2024

1. **Introducción:**

En esta memoria vamos documentar la implementación de los ejemplos propuestos en el guión, demostrando que funcionan y explicando su funcionamiento; también trataremos la implementación del Sistema domótico, su manual de usuario y algunos ejemplos de ejecución.

2. **Parte 1: Implementación de los ejemplos:**

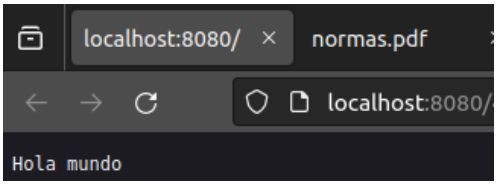
Se han implementado los 3 ejemplos propuestos.

2.1. Ejemplo 1

En este primer ejemplo,S si visitamos la dirección "http://localhost:8080/", nos aparecerá el mensaje "Hola Mundo". Básicamente en el código, se crea un servidor http y una función con dos parámetros: request y response, que mediante http muestra el mensaje cada vez que se recibe una petición, es decir una conexión.

```
import http from 'node:http';

http.createServer((request, response) => {
        console.log(request.headers);
        response.writeHead(200, {'Content-Type': 'text/plain'});
        response.write('Hola mundo');
        response.end();
    })
    .listen(8080);

console.log('Servicio HTTP iniciado');
```

helloworld.js

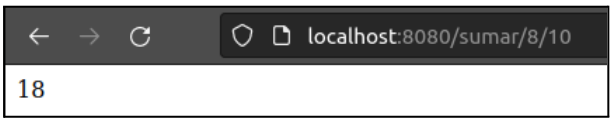


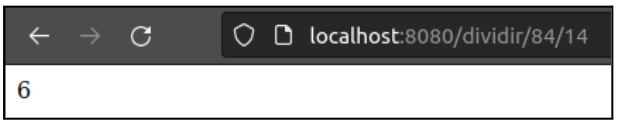
2.2. Ejemplo 2

En el segundo ejemplo, este servicio recibe peticiones REST del tipo "http://localhost:8080/sumar/8/10", mostrando en pantalla el resultado. En el código, según la url proporcionada se realizan distintas operaciones y finalmente, igual que en el ejemplo anterior, se muestra por pantalla un mensaje, en este caso el resultado numérico.

```
import http from 'node:http';

function calcular(operacion, val1, val2) {
    if (operacion == 'sumar') return val1+val2;
    else if (operacion == 'restar') return val1-val2;
    else if (operacion == 'producto') return val1*val2;
    else if (operacion == 'dividir') return val1/val2;
    else return 'Error: Parámetros no válidos';
}

http.createServer((request, response) => {
        let {url} = request;
        url = url.slice(1);
        const params = url.split('/');
        let output='';
        if (params.length >= 3) {
            const val1 = parseFloat(params[1]);
            const val2 = parseFloat(params[2]);
            const 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; charset=utf-8'});
        response.write(output);
        response.end();
    })
    .listen(8080);

console.log('Servicio HTTP iniciado');
```

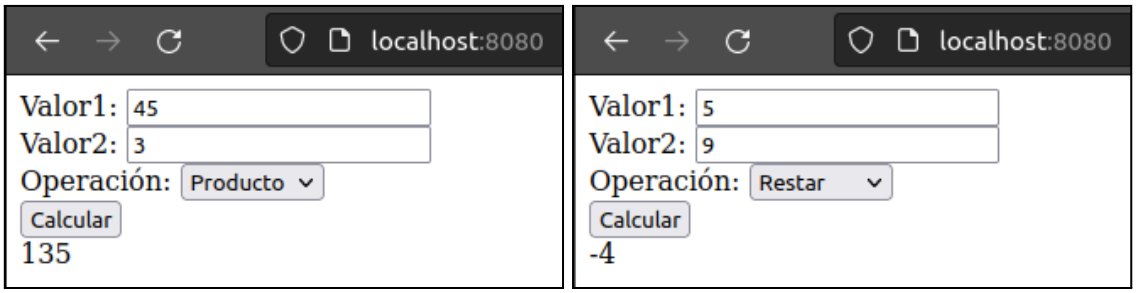
calculadora.js





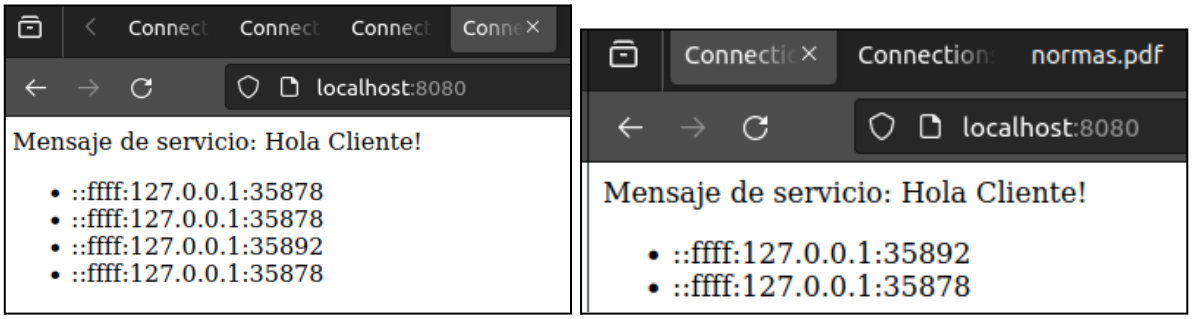
2.3. Ejemplo 3

Este ejemplo es una mejora del servicio anterior, porcionando una interfaz web para la calculadora en caso de que el usuario acceda desde el navegador a la url “http://localhost:8080”. Además se usa node:fs para entrada-salida.



2.4. Ejemplo 4

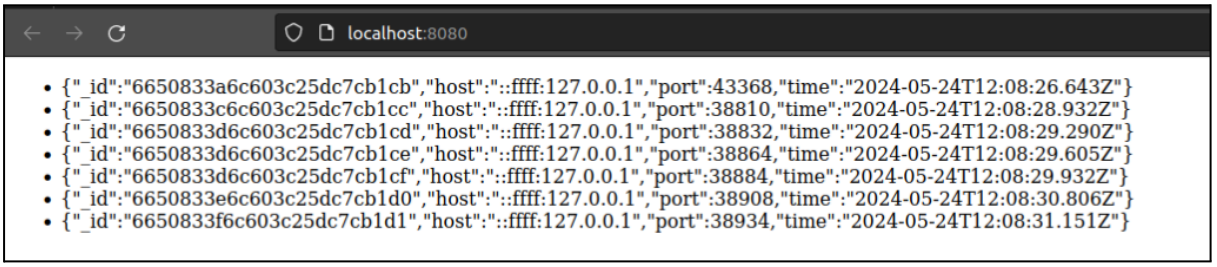
Este ejemplo muestra la implementación sobre Socket.io de un servicio que envía una notificación que contiene las direcciones de todos los clientes conectados al propio servicio. Dicha notificación se envía a todos los clientes suscritos cada vez que uno nuevo se conecta o desconecta. El envío a todos los clientes se realiza llamando a la función emit sobre el conjunto de clientes conectados. Además, cuando el servicio recibe un evento de tipo “output-evt” le envía al cliente el mensaje “Hola Cliente!”.




```
alv2311lp@alv2311lp-ASUS-TUF:~/Escritorio/DSD/practica4/codigoEjemplos$ node connections.js

Servicio Socket.io iniciado
Nueva conexión de ::ffff:127.0.0.1:35878
Nueva conexión de ::ffff:127.0.0.1:35878
Nueva conexión de ::ffff:127.0.0.1:35892
Nueva conexión de ::ffff:127.0.0.1:35878
El usuario ::ffff:127.0.0.1:35878 se va a desconectar
El usuario ::ffff:127.0.0.1:35878 se ha desconectado
El usuario ::ffff:127.0.0.1:35878 se va a desconectar
El usuario ::ffff:127.0.0.1:35878 se ha desconectado
```

2.5. Ejemplo 5

Este último ejemplo muestra la implementación de un servicio que recibe dos tipos de notificaciones mediante Socket.io: “poner” y “obtener”. El contenido del primer tipo de notificación es introducido por el servicio en la base de datos “baseDatosTest”, dentro de la colección de claves-valor “test”. Al recibir el segundo tipo de notificación, el servicio hace una consulta sobre la colección “test” en base al contenido de la propia notificación y le devuelve los resultados al cliente. Además, cuando un cliente se conecta, el servicio le devuelve su dirección de conexión.



3. Parte 2: Sistema domótico

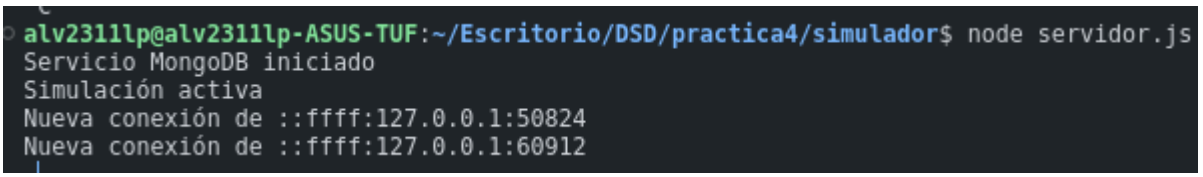
En este ejercicio se ha implementado un sistema domótico con 3 sensores, luminosidad, temperatura y humo, con 3 actuadores: persiana, AC y sistema de incendios. Cuenta con un servidor que muestra las páginas y sirve como medio para la comunicación entre los sensores y el usuario, es además, capaz de procesar la información de los sensores para así, actuar en consecuencia enviando alarmas o activando los sistemas pertinentes.

Nuestra implementación consta de varios archivos:

- usuario.html: implementa la interfaz que ve el supuesto usuario, con lo valores de los sensores y actuadores, los botones para activar o desactivar los mismos, historial de cambios y un apartado para las alarmas.
- sensores.html: implementa la interfaz para modificar los valores de los sensores, muestra los valores de los mismos y de los actuadores, y cuenta con campos de entrada para modificarlos.
- servidor.js: Es el archivo principal que lanza los archivos html, conecta con la base de datos e implementa la lógica del sistema (agente).
- Archivos de estilo CSS: para dar un estilo más amigable a la interfaz del navegador

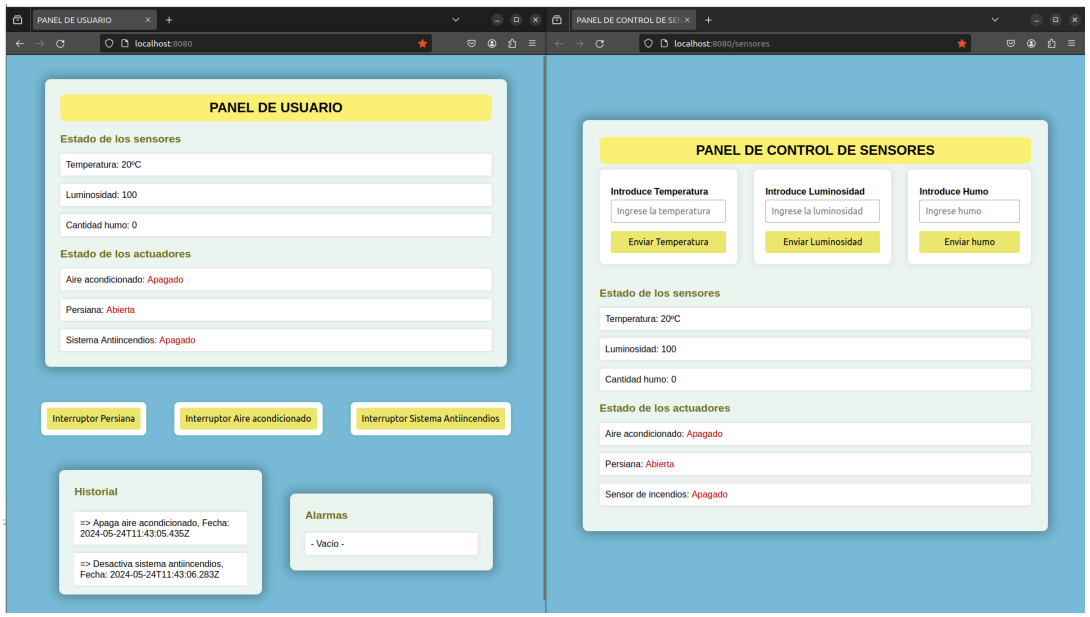
3.1 Manual de usuario

Vamos a comentar cómo usar el ejercicio implementado. Primero, hemos de ejecutar el servidor. Podemos hacerlo con el comando “*node servidor.js*” .



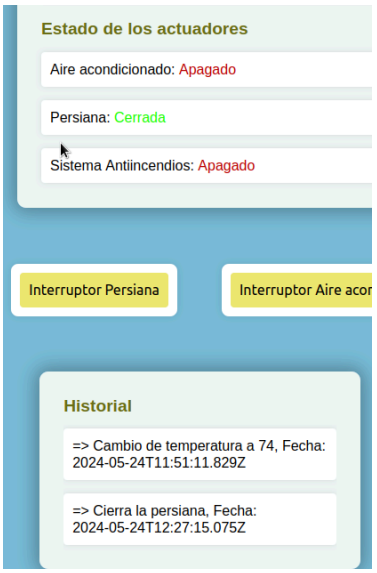
Se mostrará por la terminal las conexiones de las diferentes páginas que hemos implementado, pudiendo tener más de una ventana de usuario.

Tras lanzar el servidor, podemos conectarnos mediante un navegador a la página de usuario y a la de sensores. Esto es en localhost:8080 y localhost:8080/sensores.



Aquí como comentamos antes podemos desde el usuario, alternar el estado de las actuadores, pulsando en los botones que se muestran. Esto creará nuevas entradas en el historial, el cual muestra los últimos 10 eventos del sistema.

Por otra parte tenemos el panel de los sensores, en el cual podemos modificar los valores de los mismos y así ver la lógica implementada, que la comentamos en el siguiente apartado.
En los formularios visibles podemos introducir un número y clicando en enviar, se actualizará tanto en panel de sensores como el de usuario.
Aplicando así, según los umbrales, la activación por parte del servidor de los actuadores correspondientes y de las alarmas.



3.2 Lógica del sistema

La lógica implementada, se ha realizado en el servidor, en concreto en la función agente:

```
//////////////////////////////// Funcion Agente ////////////////////////////////
function agente(){
    var resultado = {mensajes: []};

    if( temperaturaAct > umbralTempMax){
        aireAcondicionado = true;
        resultado.mensajes.push("ALARMA: LÍMITE MÁXIMO DE TEMPERATURA");
        console.log("** Agente: se sobrepasa el limite máximo de temperatura **");
    }
    if( temperaturaAct < umbralTempMin){
        aireAcondicionado = false;
        resultado.mensajes.push("ALARMA: LÍMITE MÍNIMO DE TEMPERATURA");
        console.log("** Agente: se sobrepasa el limite mínimo de temperatura **");
    }

    if( luminosidadAct > umbralLumMax ){
        persiana = true;
        resultado.mensajes.push("ALARMA: LÍMITE MÁXIMO DE LUMINOSIDAD");
        console.log("** Agente: se sobrepasa el limite máximo de luminosidad **");
    }
    if( luminosidadAct < umbralLumMin ){
        persiana = false;
        resultado.mensajes.push("ALARMA: LÍMITE MÍNIMO DE LUMINOSIDAD");
        console.log("** Agente: se sobrepasa el limite mínimo de luminosidad **");
    }

    if( humoAct > umbralHumoMax ){
        persiana = false;
        resultado.mensajes.push("ALARMA: LÍMITE 1 MAXIMO DE HUMO");
        console.log("** Agente: se sobrepasa el limite 1 de humo **");

    }
```

De esta forma, cuando iniciamos la aplicación, tenemos uno valores predeterminados que son :

```
var luminosidadAct = 100;
var temperaturaAct = 20;
var humoAct = 0;
```

Estos valores, irán cambiando a medida que los modifiquemos en el panel de sensores, de forma que nuestro agente según estos umbrales:

```
var umbralTempMax = 35;
var umbralLumMax = 200;
var umbralHumoMax = 20;
var umbralHumoMax2 = 50;
var umbralTempMin = 10;
var umbralLumMin = 50;
var persiana = false;
var aireAcondicionado = false;
var antiincendios = false;
```

,modificará los actuadores y enviará alarmas de la siguiente manera:

- Si la temperatura es superior al umbral max, activa el AC.
- Si la temperatura es inferior al umbral min, desactiva el AC.
- Si la temperatura está entre ambos umbrales, no se modifica el estado del AC.

Para la luminosidad ocurre de la misma manera junto con la persiana. Sin embargo para el humo contemplamos un caso distinto:

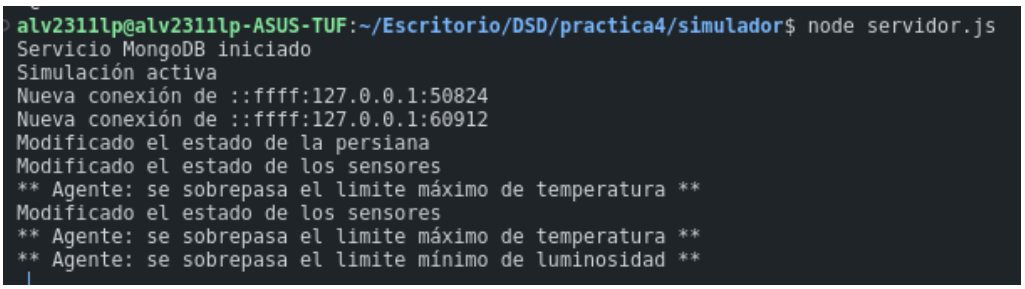
- Si el humo supera el umbral 1, se abre la persiana independientemente de la luminosidad.
- Si el humo supera el umbral 2, se activa además el sistema de incendios.
- Y si el humo es inferior a ambos se apaga el sistema de incendios y la persiana se modificará entonces por la luminosidad.

3.3 Otras consideraciones

Se ha empleado socket.io para que nodos y usuarios cooperen entre sí (descubrimiento, actualización de datos).

Se ha empleado una base de datos de mongodb para mantener el historial de eventos.

Durante la ejecución también se muestran mensajes por la terminal del servidor, estos son, notificando cambios, alarmas y la lógica mediante el agente.



4. **Anexos:**

Junto con esta memoria se incluye todo el código fuente.