



**ETSIIT**

Escuela Técnica Superior  
de Ingenierías Informática  
y de Telecomunicación



**UNIVERSIDAD  
DE GRANADA**

**Desarrollo de Sistemas Distribuidos**

**AUTOR: Jaime Parra Jimenéz**

# ÍNDICE

<b>ÍNDICE</b>	<b>2</b>
<b>1. INTRODUCCIÓN</b>	<b>3</b>
<b>2. EJEMPLOS</b>	<b>3</b>
2.1 EJEMPLO 1	3
2.2 EJEMPLO 2	3
2.3 EJEMPLO 3	4
2.4 EJEMPLO 4	4
2.5 EJEMPLO 5	5
<b>3. EJERCICIO</b>	<b>6</b>
3.1 FUNCIONALIDAD AÑADIDA	8
3.2 EJECUCIÓN DEL PROGRAMA	8

# 1. INTRODUCCIÓN

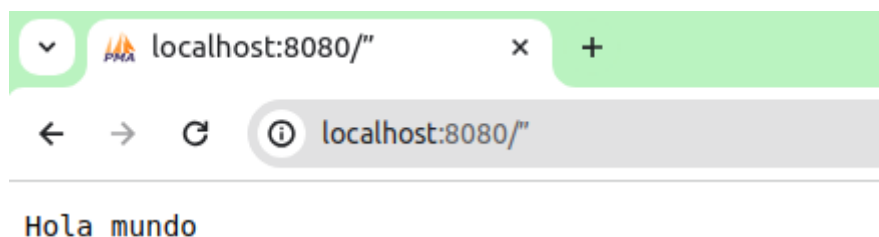
Node.js es un entorno de tiempo de ejecución JavaScript multiplataforma y de código abierto y que se ejecuta en el motor V8 JavaScript y ejecuta código JavaScript fuera de un navegador web.

Node.js permite a los desarrolladores utilizar JavaScript para escribir herramientas de línea de comandos y para el scripting del lado del servidor y tiene una arquitectura impulsada por eventos capaz de la E/S asíncrona.

## 2. EJEMPLOS

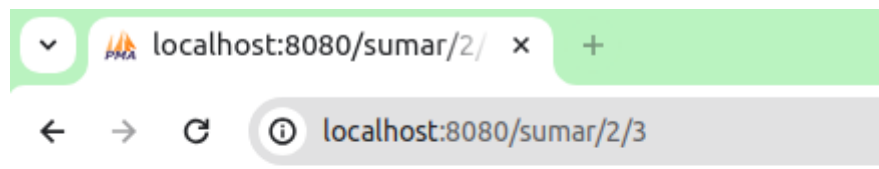
### 2.1 EJEMPLO 1

En este ejemplo se crea un servidor HTTP básico utilizando Node.js, se manejan solicitudes HTTP entrantes y se envían respuesta el cliente. Al ejecutar este código y acceder a la dirección 'http://localhost:8080/', se mostrará el mensaje 'Hola Mundo' en el navegador web.



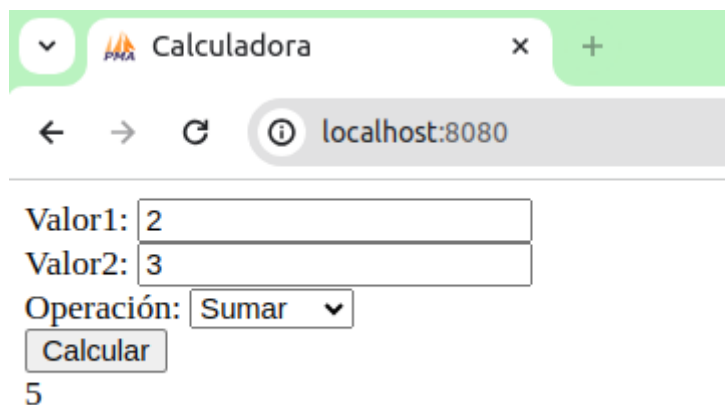
### 2.2 EJEMPLO 2

En este ejemplo se crea un servidor HTTP en Node.js que implementa una calculadora distribuida usando la interfaz REST. El servidor acepta solicitudes HTTP donde se le pasa los dos valores y la operación. El resultado de la operación se devuelve como respuesta al cliente.



## 2.3 EJEMPLO 3

Este ejemplo proporciona una interfaz web para una calculadora distribuida utilizando Node.js. Los usuarios pueden acceder a esta interfaz desde su navegador y realizar operaciones matemáticas simples. La lectura del archivo HTML y el cálculo de las operaciones se realizan de manera asíncrona para mejorar la eficiencia del servidor.



Valor1: 2

Valor2: 3

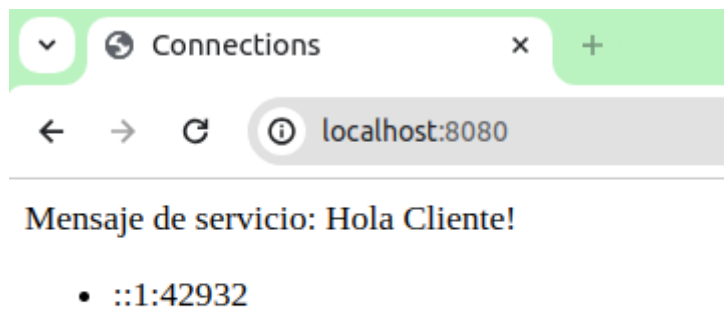
Operación: Sumar ▼

Calcular

5

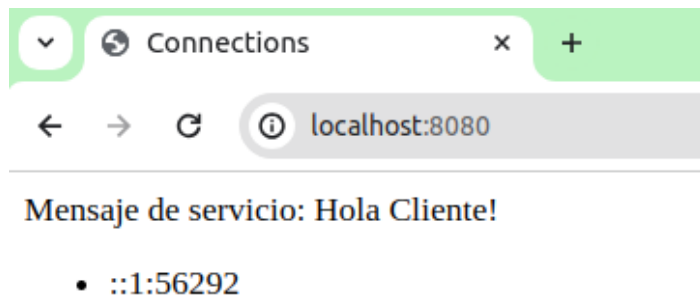
## 2.4 EJEMPLO 4

Este ejemplo muestra cómo implementar un servicio de Socket.io para la comunicación bidireccional entre el servidor y los clientes, y cómo proporcionar una interfaz web para interactuar con el servicio. Los clientes pueden conectarse al servidor, enviar y recibir eventos, y recibir notificaciones sobre cambios en la lista de usuarios conectados.



Mensaje de servicio: Hola Cliente!

- ::1:42932

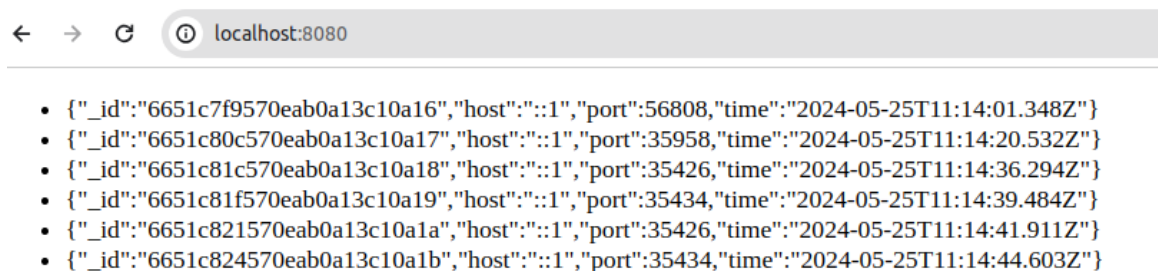


```
jaiame@jaiame-MSI:~/Escritorio/3AÑO/2cuatri/DSD/P4-DSD$ node connections.js
Servicio Socket.io iniciado
Nueva conexión de ::1:42932
El usuario ::1:42932 se va a desconectar
El usuario ::1:42932 se ha desconectado
Nueva conexión de ::1:56292
```

Se puede apreciar cómo se va actualizando el usuario en la web y en la terminal aparecen los clientes que se conectan y desconectan.

## 2.5 EJEMPLO 5

Este ejemplo crea un servicio que utiliza Socket.io para interactuar con una base de datos MongoDB, permitiendo a los clientes almacenar y recuperar datos de manera dinámica.



En esta foto se puede apreciar como se guarda cada usuario que se conecta. Si volvemos a acceder a la página aparecerá otro usuario.



Aquí tenemos una foto de como se vería la página con la que actúa el usuario.

Se puede apreciar como nos informa de los datos de los sensores, nos permite modificar los actuadores, nos muestra los datos recogidos por los sensores, las alertas del agente y los clientes conectados.

Para la decoración de la página se ha usado css para darle un mejor aspecto a esta.

- sensores.html: En esta página es donde se recogen los datos recogidos por los sensores. Estos datos se pasan al servidor para que los distribuya con todos los usuarios. También nos muestra si los actuadores están apagados o encendidos.

SENSORES

Temperatura: 1

Luminosidad: 32

Humo: 45

Enviar

Estado de los actuadores

Aire: OFF

Persiana: Abierta

Detector de humo: ON

Así se vería la página de los sensores, como podemos apreciar es muy básica contando con un formulario para meter los valores de los sensores. Este formulario simula los valores que tomaría los sensores si fuesen reales. Debajo del formulario podemos ver el estado de nuestro actuadores.

Al igual que con la página de cliente.html se ha usado css para mejorar su aspecto.

- servidor.js: Este archivo es el encargado de distribuir la información entre los sensores y los clientes. Primero se crea un servidor HTTP que te dirige a usuario.html o sensores.html dependiendo de la URL que se le pase. A continuación, se conecta con la base de datos para ir almacenando los valores deseados. Una vez conectado a la base de datos se va pasando y recibiendo información de los archivos html de los clientes conectados. Mediante io.sockets.on se recibe la información y con io.sockets.emit se manda la información. Cada mensaje de respuesta o recibido va asociado a un nombre para poder diferenciar los mensajes y que no se pierda información.

Cuando se conecta el servidor se imprime en la terminal la URL de nuestro servidor con la que el usuario puede conectarse.

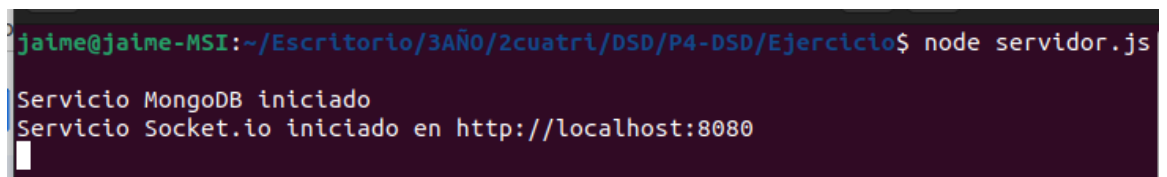
### 3.1 FUNCIONALIDAD AÑADIDA

Al ejemplo básico he añadido alguna funcionalidad para interactuar con el programa. Primeramente, he añadido un nuevo sensor de humo, el cual detecta la cantidad de humo que hay. Además he añadido un nuevo actuador, un sensor de humo, el cual interactuará con el nuevo sensor. El agente es el encargado de cambiar los actuadores cuando se supera o no alcanza un valor determinado por el servidor. Cuando el agente entra en acción he implementado que mande una alerta a los usuarios informando de que ha cambiado el actuador. También, he implementado que antes de mandar la alerta compruebe si el actuador ya se encuentra en el estado al que lo iba a cambiar, si es así, el agente no mandará ninguna alerta. Por ejemplo, si el aire está encendido y la temperatura supera el límite para que el agente actúe para encenderlo no se mandaría ninguna alerta a los usuarios, en cambio, si se encuentra apagado y supera el límite se encendería y aparte se mandaría una alerta.

### 3.2 EJECUCIÓN DEL PROGRAMA

Para poder ejecutar el programa deberemos ir a la carpeta donde se encuentra nuestro archivo servidor.js. Una vez allí, deberemos poner en la terminal el siguiente comando:

```
node servidor.js
```



```
jaime@jaime-MSI:~/Escritorio/3AÑO/2cuatri/DSD/P4-DSD/Ejercicio$ node servidor.js
Servicio MongoDB iniciado
Servicio Socket.io iniciado en http://localhost:8080
```

Al ejecutar este comando nos aparece este mensaje en la terminal, esto quiere decir que se ha conectado con éxito. Para abrir nuestra página de los sensores debemos acceder a ella pinchando en el enlace que aparece en la terminal o escribiendo en nuestro navegador <http://localhost:8080>. Para acceder a nuestra página de usuarios deberemos añadir a la URL /cliente quedando la URL de esta manera: <http://localhost:8080/usuario>.

Con estos comandos podremos lanzar nuestro programa con el que podremos interactuar sin problemas.