

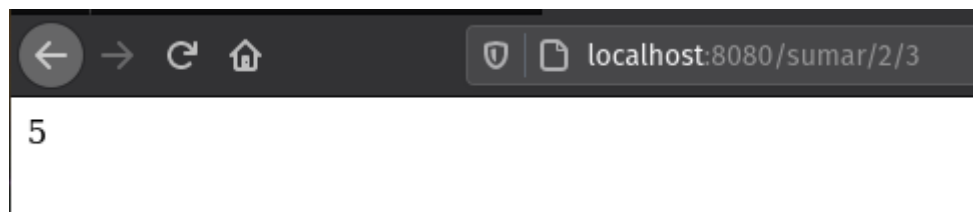
Hola mundo

```
raulrguez@pop-os:~/Escritorio/DSD_P4/codigos-guion2020$ node helloworld.js
Servicio HTTP iniciado
{
  host: 'localhost:8080',
  'user-agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:85.0) Gecko/20100101 Firefox/85.0',
  accept: '*/*',
  'accept-language': 'es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3',
  'accept-encoding': 'gzip, deflate',
  connection: 'keep-alive',
  referer: 'http://localhost:8080/'
}
```

Tal y como se explica en el documento de la práctica, al ejecutarlo y entrar en la dirección '<http://localhost:8080>', observamos que se muestra el 'hola mundo' que hemos introducido en el archivo. La explicación del código viene bien detallada en dicho documento, pero quiero destacar la parte donde nos explican que, cuando se llega al final del código, el servicio no termina su ejecución, puesto que en Node.js la mayoría de las operaciones no son bloqueantes. Aún así, cabe destacar que, se impide la finalización de un programa mientras existan puertos del sistema operativo en uso.

3. Ejemplo 2: Calculadora distribuida usando interfaz REST

En este caso tenemos dos opciones, la primera de ellas es ejecutar el archivo 'calculadora.js'. Este archivo recibe peticiones REST, tal y como comentamos en el título, siendo estas peticiones del tipo "http://localhost:8080/sumar/2/3". Es decir, pasando en la url el tipo de operación básica y los 2 operandos, el programa nos devolverá el resultado de dicha operación:



Por otro lado, podemos ejecutar el archivo 'calculadora-web.js', que realiza lo mismo que el anterior, pero esta vez con una interfaz visual. Esta interfaz muestra un formulario cuyo envío de datos se realiza mediante una función JavaScript. Esta obtiene los valores introducidos por el usuario en el formulario, crea una petición tipo REST y finalmente envía la petición al servicio de manera asíncrona (con XMLHttpRequest).

Valor1: 2

Valor2: 3

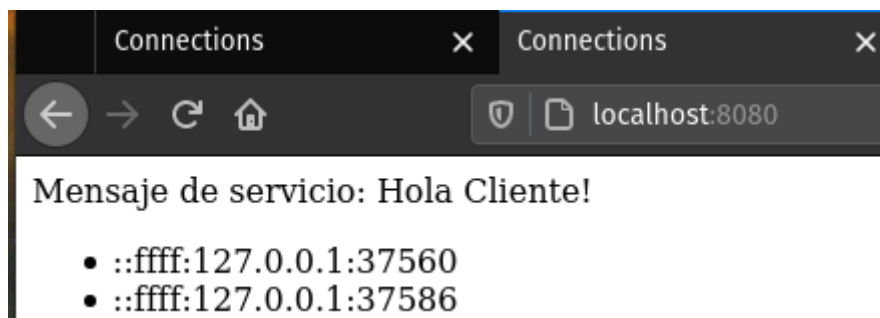
Operación: Sumar ▼

Calcular

5

4. Ejemplo 3: Aplicaciones en Tiempo Real con Socket.io

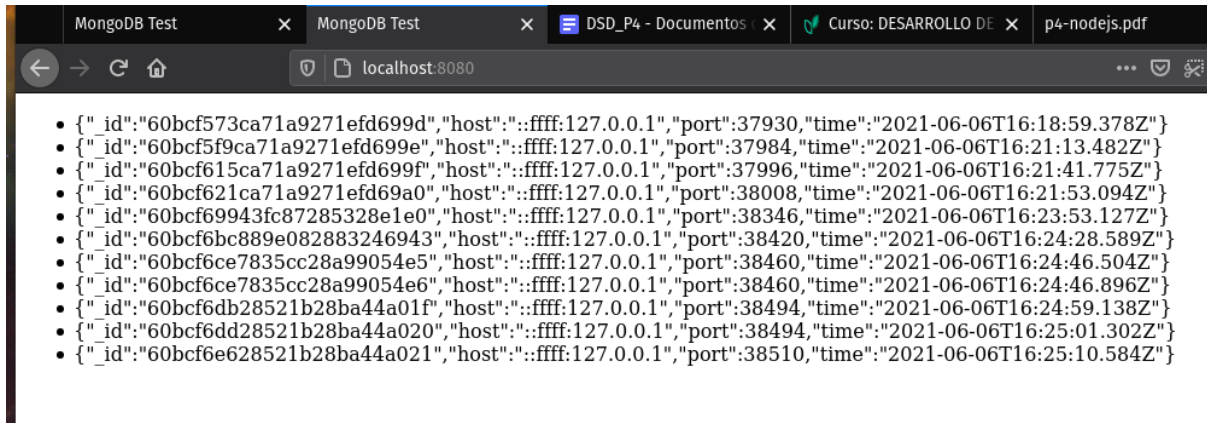
El siguiente ejemplo muestra la implementación sobre Socket.io de un servicio que envía una notificación con las direcciones de todos los clientes conectados al propio servicio. Para realizar dicho envío, se llama a la función emit sobre el array de clientes conectados 'io.sockets.emit(...)'. Además, cabe destacar, que dicha notificación (acompañada de un mensaje: Hola Cliente!) se envía a todos los clientes suscritos cada vez que un nuevo cliente se conecta o desconecta.



5. Ejemplo 4: Uso de MongoDB desde Node.js

El siguiente ejemplo se emplea a modo de introducción para emplear MongoDB con lo aprendido en los anteriores ejemplos. En este caso, la implementación muestra un servicio que recibe dos tipos de notificaciones mediante Socket.io: "poner" y "obtener". El contenido de 'poner' es introducido por el servicio en la base de datos "mibd", dentro de la colección de claves-valor "test". Y al recibir el contenido de 'obtener', el servicio hace una consulta sobre la base de datos "mibd" 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.



```
{ "_id": "60bcf573ca71a9271efd699d", "host": "::ffff:127.0.0.1", "port": 37930, "time": "2021-06-06T16:18:59.378Z" }
{ "_id": "60bcf5f9ca71a9271efd699e", "host": "::ffff:127.0.0.1", "port": 37984, "time": "2021-06-06T16:21:13.482Z" }
{ "_id": "60bcf615ca71a9271efd699f", "host": "::ffff:127.0.0.1", "port": 37996, "time": "2021-06-06T16:21:41.775Z" }
{ "_id": "60bcf621ca71a9271efd69a0", "host": "::ffff:127.0.0.1", "port": 38008, "time": "2021-06-06T16:21:53.094Z" }
{ "_id": "60bcf69943fc87285328e1e0", "host": "::ffff:127.0.0.1", "port": 38346, "time": "2021-06-06T16:23:53.127Z" }
{ "_id": "60bcf6bc889e082883246943", "host": "::ffff:127.0.0.1", "port": 38420, "time": "2021-06-06T16:24:28.589Z" }
{ "_id": "60bcf6ce7835cc28a99054e5", "host": "::ffff:127.0.0.1", "port": 38460, "time": "2021-06-06T16:24:46.504Z" }
{ "_id": "60bcf6ce7835cc28a99054e6", "host": "::ffff:127.0.0.1", "port": 38460, "time": "2021-06-06T16:24:46.896Z" }
{ "_id": "60bcf6db28521b28ba44a01f", "host": "::ffff:127.0.0.1", "port": 38494, "time": "2021-06-06T16:24:59.138Z" }
{ "_id": "60bcf6dd28521b28ba44a020", "host": "::ffff:127.0.0.1", "port": 38494, "time": "2021-06-06T16:25:01.302Z" }
{ "_id": "60bcf6e628521b28ba44a021", "host": "::ffff:127.0.0.1", "port": 38510, "time": "2021-06-06T16:25:10.584Z" }
```

6. Sistema domótico básico

Para este ejercicio he recreado el escenario de un sistema domótico básico compuesto de numerosos sensores (luminosidad, temperatura, humedad, viento), actuadores (motor persiana, ventana, sistema de Aire/Acondicionado, calefacción, humidificador), y un servidor que sirve páginas para mostrar el estado y actuar sobre los elementos de la vivienda, además de incluir un agente capaz de notificar alarmas y tomar decisiones básicas.

A continuación voy a proceder a explicar parte por parte cada uno de los componentes de dicho sistema:

- Sensores

En mi implementación he querido añadir un par de sensores más a parte de los básicos que pedían (temperatura y luminosidad). Por ello, el primer sensor que he añadido ha sido el de humedad, el cual se encargará de detectar y regular el nivel de humedad, de en este caso, mi habitación. Con dicho sensor, también he añadido un actuador más, como es el caso del humidificador. Este se emplea para aumentar el nivel de humedad en el aire, y hace un equipo ideal si se emplea a la misma vez que el aire acondicionado. Por otro lado, el último sensor extra que he añadido ha sido el del viento, pero este tiene un aspecto especial. A diferencia del resto, los cuales, les asigno manualmente un valor por defecto, y unos límites tanto mínimo como máximos, el viento lo extraigo directamente de la página OpenWeather, la cual me da en tiempo real, la velocidad del viento en Granada. Para conseguir esto, lo primero que hice fue registrarme en la web. Una vez registrado, busque el tutorial indicado para ver cómo podía extraer los datos y encontré el siguiente:

- https://manuais.iessanclemente.net/index.php/Node.js_y_Web_de_Informaci%C3%B3n_meteorol%C3%B3gica

En él se explica detalladamente cómo, haciendo uso de 'request' y de la Api key que nos proporciona la web al registrarnos, podemos conseguir obtener los datos en tiempo real.

Cabe destacar que también añadí un actuador sobre este sensor, que sería en este caso, una ventana. La cual si la velocidad del viento es excesiva, el agente se dispondrá a cerrarla. Por último, también añadí calefacción, como un nuevo actuador para la temperatura.

- Servidor

El servidor es el encargado de proporcionar por un lado el formulario en el que se modificarán los valores de los sensores. Por otro lado, también mostrará dichos valores, los estados de los actuadores, las respuestas del agente (las alarmas sobre si alguno de los valores ha excedido los límites a parte de los avisos sobre las modificaciones sobre los actuadores), un registro de todas las modificaciones que se realicen a los sensores, y por último, mantendrá las suscripciones, de los usuarios que se encuentren accediendo al sistema.

En el archivo 'servidor.js' hay tres partes claramente diferenciadas. En primer lugar tenemos la declaración de los valores de los sensores y sus límites, junto con la función 'agente()'. En dicha función dependiendo de los valores de los sensores, y los estados de los actuadores, se guardan en un array, las alarmas que mostrará el agente. En segundo lugar tenemos la creación del servidor, la cual, no tiene mucha complicación ya que es muy parecida a las creaciones que hemos hecho para los ejemplos. Y por último, tenemos la parte de MongoDB, en la cual, realizamos diversas funcionalidades. Por un lado tenemos el registro de usuarios, tal y como lo hicimos en el ejemplo número 3. A continuación tenemos diversos métodos que se encuentran escuchando, o mejor dicho, buscando peticiones (emit) para su uso. Entre ellos, tenemos un grupo los cuales esperan una petición para apagar alguno de los actuadores y otro para captar los valores introducidos en el formulario de la web. Por otro lado tenemos los métodos que envían la petición (emit) como son, uno para devolver el listado de usuarios, otro para devolver el registro de modificaciones, y por último para devolver los estados, valores y respuestas del agente,

Sensores sistema domótico

Simulador de valores

Valor a simular:

Sensor a simular: Temperatura ▼

Simular

Valores de los sensores

Temperatura: 25°C
Luminosidad: 10%
Persiana: cerrada
Estado Aire Acondicionado: apagado
Estado del agente domótico: encendido
Humedad: 50%
Estado Calefacción: apagado
Estado Humidificador: apagado
Estado ventana: abierta
Velocidad viento: 3.13 km/h

Respuesta del agente

Agente preparado para realizar sus funciones

Registro de la modificacion de valores

"El sensor de luminosidad cambia su valor a 10 en la fecha: 2021-06-04T16:31:13.459Z"
"El sensor de luminosidad cambia su valor a 10 en la fecha: 2021-06-04T16:31:14.920Z"
"El sensor de luminosidad cambia su valor a 10 en la

Usuarios activos en el sistema

"Nuevo usuario conectado en el puerto 38898"
"Nuevo usuario conectado en el puerto 39406"
"Nuevo usuario conectado en el puerto 39424"
"Nuevo usuario conectado en el puerto 39442"
"Nuevo usuario conectado en el puerto 39462"

Página usuario

Además tal y como se aprecia en la captura de pantalla, tenemos un botón que nos redirige a la página de usuario. En dicha página a parte de mostrar los valores de los sensores y el estado de los actuadores, además de las respuestas del agente, también se da la posibilidad al usuario de modificar los actuadores manualmente, es decir, apagar el aire acondicionado por ejemplo.

Por último, cabe destacar que en los archivos 'servidor.html' y 'usuario.html', a parte del html necesario para mostrar toda la información, se encuentran en un script, los diversos métodos socket.emit, que se encargan de mandar las peticiones a todos los métodos que se encontraban escuchando en el servidor.js. Destacando que, en el primero de estos archivos empleamos los métodos para obtener los valores de los sensores, estado de los actuadores, registro de modificaciones y registro de usuarios. Mientras que en el segundo archivo nos encontramos con los valores de los sensores, estado de los actuadores y todos los métodos para apagar/encender los actuadores.

- Ejemplo de ejecución

Una vez explicado la implementación, voy a mostrar un pequeño caso de uso con sus respectivas capturas de pantalla. En primer lugar ejecutamos el servidor con: '\$node servidor.js'

```
raulrguez@pop-os:~/Escritorio/DSD_P4/DSD-master(1)/DSD-master/P4/S2$ node servidor.js
El servicio HTTP se ha iniciado
```

A continuación, abrimos el navegador y entramos en '<http://localhost:8080/>', esto nos mostrará la página de inicio del servidor:

Sensores sistema domótico

Simulador de valores

Valor a simular:

Sensor a simular: Temperatura ▾

Simular

Valores de los sensores

Temperatura: 25°C
Luminosidad: 10%
Persiana: cerrada
Estado Aire Acondicionado: apagado
Estado del agente domótico: encendido
Humedad: 50%
Estado Calefacción: apagado
Estado Humidificador: apagado
Estado ventana: abierta
Velocidad viento: 2.91 km/h

Página usuario

Respuesta del agente

Agente preparado para realizar sus funciones

Registro de la modificacion de valores

"El sensor de luminosidad cambia su valor a 10 en la fecha: 2021-06-04T16:31:13.459Z"
"El sensor de luminosidad cambia su valor a 10 en la fecha: 2021-06-04T16:31:14.920Z"
"El sensor de luminosidad cambia su valor a 10 en la

Usuarios activos en el sistema

"Nuevo usuario conectado en el puerto 40626"
"Nuevo usuario conectado en el puerto 40630"
"Nuevo usuario conectado en el puerto 40658"

Vamos a simular que la temperatura de la habitación aumenta en exceso para saber cómo reaccionará el agente:

Simulador de valores

Valor a simular:

Sensor a simular:

Temperatura

Simular

Valores de los sensores

Temperatura: 40°C
Luminosidad: 10%
Persiana: cerrada
Estado Aire Acondicionado: encendido
Estado del agente domótico: encendido
Humedad: 50%
Estado Calefacción: apagado
Estado Humidificador: apagado
Estado ventana: cerrada
Velocidad viento: 2.91 km/h

Página usuario

Respuesta del agente

La temperatura ha superado el valor máx permitido, el agente ha encendido el aire acondicionado ,El agente ha encendido el humidificador porque el aire acondicionado se ha encendido ,Hay algun dispositivo regulador de temperatura en uso, el agente ha cerrado la ventana

El agente nos avisa de que va a encender el humidificador y el aire acondicionado porque hace mucho calor en la habitación. Ahora vamos a abrir una página de usuario y apagar manualmente el humidificador:

Página usuario

Valores de los sensores

Temperatura: 40°C
Luminosidad: 10%
Persiana: cerrada
Estado Aire Acondicionado: encendido
Estado del agente domótico: encendido
Humedad: 50%
Estado Calefacción: apagado
Estado Humidificador: encendido
Estado ventana: cerrada
Velocidad viento: 2.91 km/h

Acciones manuales

ON/OFF Aire Acondicionado

ON/OFF Persiana

ON/OFF Ventana

ON/OFF Humidificador

ON/OFF Calefacción

ON/OFF Agente Domótico

Respuesta del agente

El usuario ha encendido el humidificador de manera manual

Apreciamos que la notificación también llega al servidor, y se modifica:

Sensores sistema domótico

Simulador de valores

Valor a simular:

Sensor a simular:

Temperatura

Simular

Valores de los sensores

Temperatura: 40°C
Luminosidad: 10%
Persiana: cerrada
Estado Aire Acondicionado: encendido
Estado del agente domótico: encendido
Humedad: 50%
Estado Calefacción: apagado
Estado Humidificador: encendido
Estado ventana: cerrada
Velocidad viento: 2.91 km/h

Página usuario

Respuesta del agente

El usuario ha encendido el humidificador de manera manual