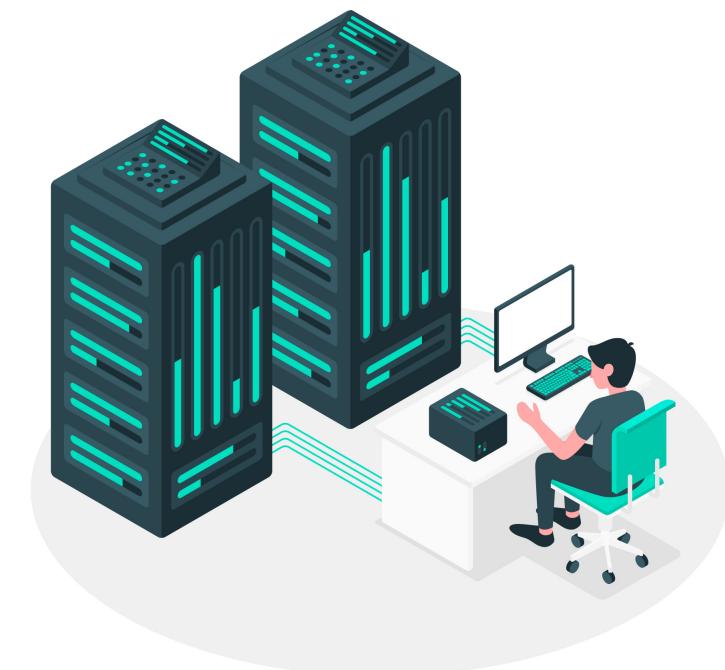


DESARROLLO DE SISTEMAS DISTRIBUIDOS
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Memoria Práctica 4

Mario López González

24 de mayo de 2022

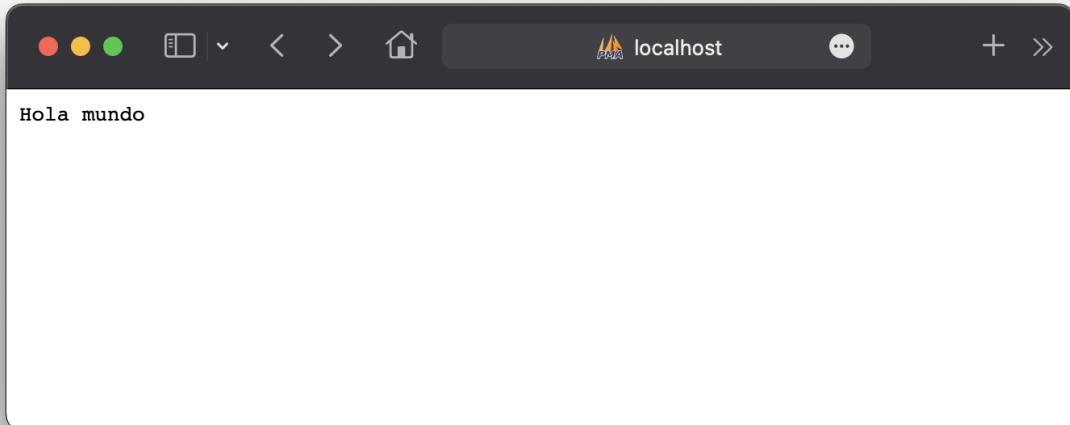


Índice

1 Practica 4: Node.JS - Ejemplos	3
1.1 Ejemplo 1: helloworld.js	3
1.2 Ejemplo 2: calculadora.js	3
1.3 Ejemplo 3: calculadora-web.js	4
1.4 Ejemplo 4: connections.js	6
1.5 Ejemplo 5: mongo-test.js	7
2 Practica 4: Node.JS - IOT Simulator	8
2.1 httpServer.js	8
2.2 dbServer.js	11
2.3 socketio.js	13
2.4 index.js	13
2.5 Notificación del cambio de sensores y/o dispositivos	14
2.6 Ejemplos de ejecución	15

1. Practica 4: Node.JS - Ejemplos

1.1. Ejemplo 1: helloworld.js

A screenshot of a terminal window. The title bar says "~/Desktop/UGR/Tercero/Segundo Cuatrimestre/DSD/P4/ejemplos". The terminal displays the following JSON object representing HTTP request headers:

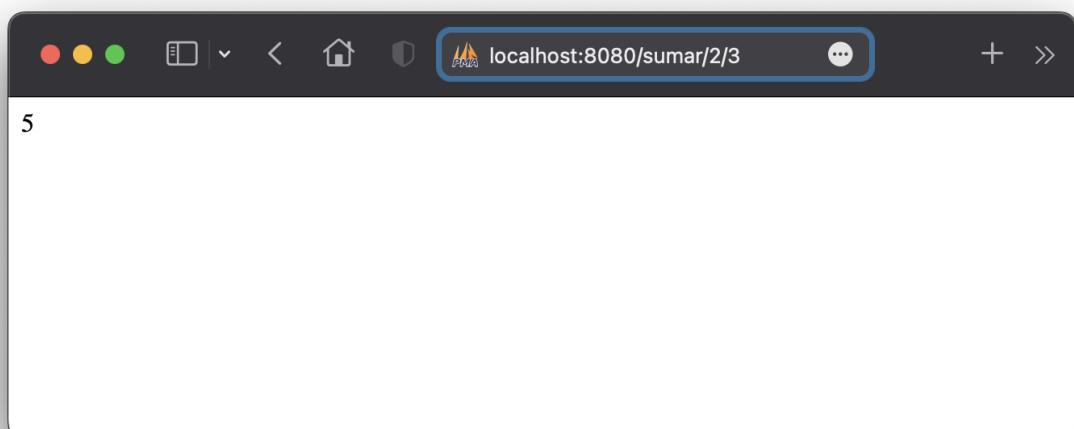
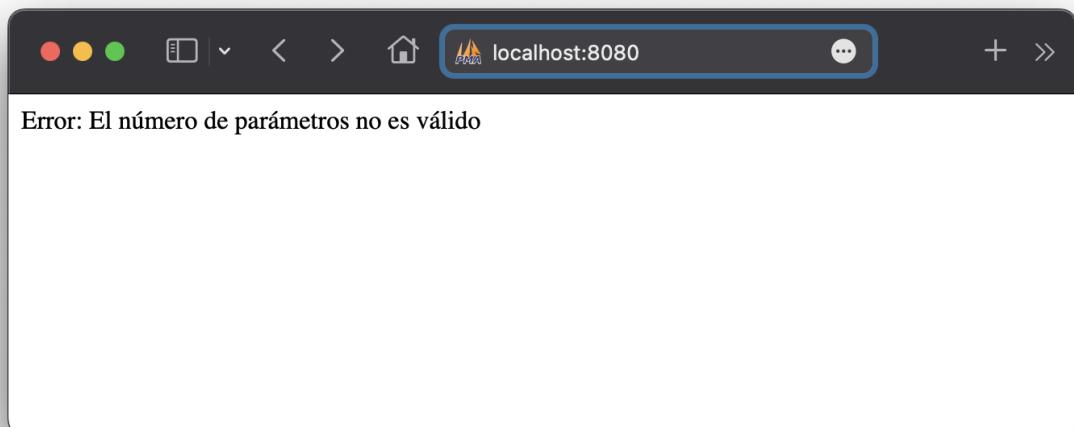
```
{  "accept-language": "es-ES,es;q=0.9",  "referer": "http://localhost:8080/",  "accept-encoding": "gzip, deflate"}{  "host": "localhost:8080",  "connection": "keep-alive",  "accept": "*/*",  "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.4 Safari/605.1.15",  "accept-language": "es-ES,es;q=0.9",  "referer": "http://localhost:8080/",  "accept-encoding": "gzip, deflate"}
```

Es un programa sencillo en el que se crea un servidor http e imprime por pantalla Hola mundo. En la terminal se imprime con un request de las cabeceras.

1.2. Ejemplo 2: calculadora.js

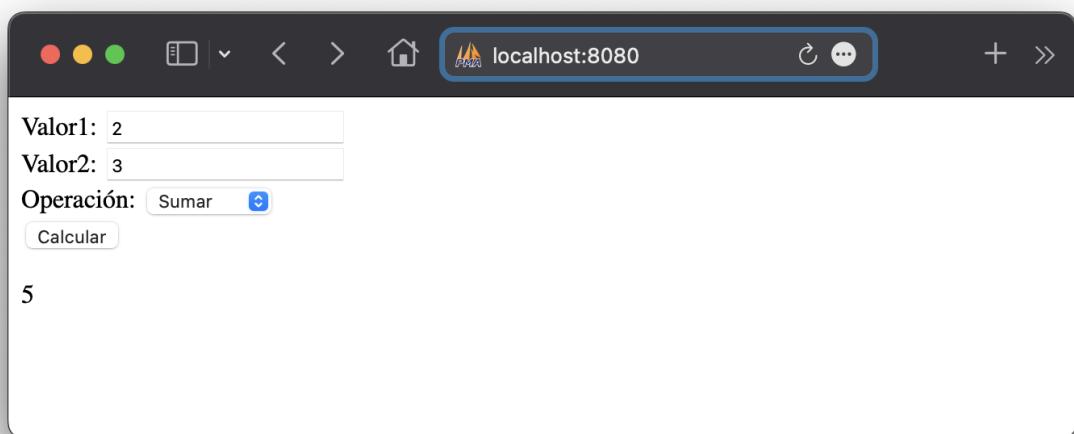
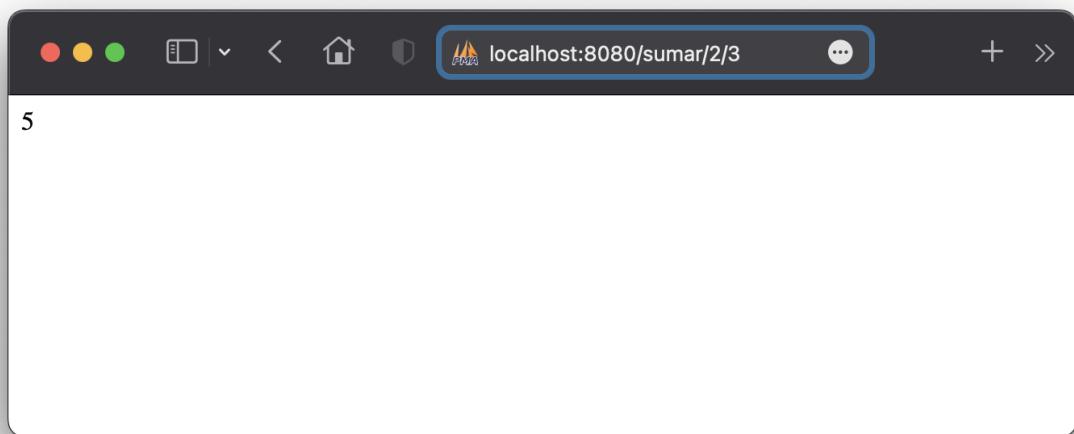
Este ejemplo obtiene la url introducida por el usuario y comprueba que sea de al menos tres parámetros separados por / en formato localhost:8080/<operacion>/<operando>/<operando>. Guarda cada parámetro en una variable distinta y lo envía a una función en la que devuelve el resultado de

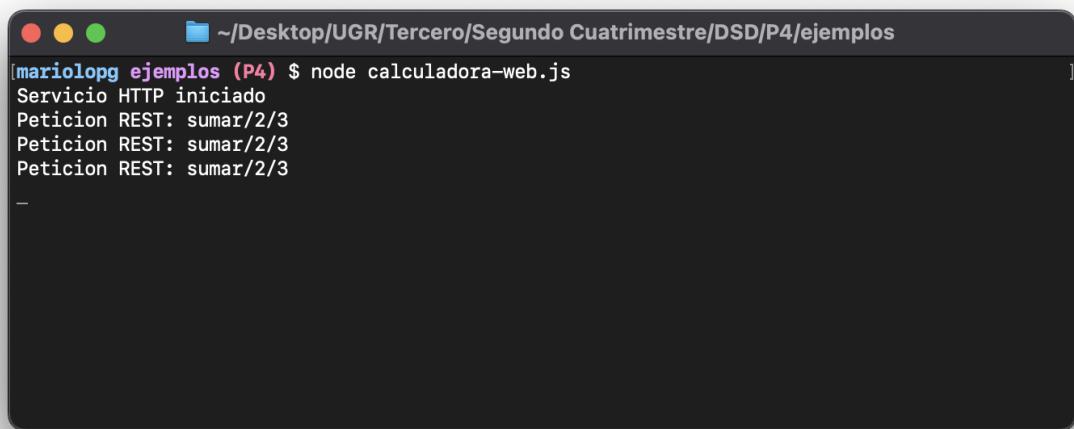
la operación según sea sumar, restar, dividir o multiplicar, en caso contrario muestra un mensaje de error. Por último lo imprime por pantalla. En caso de que el número de parámetros sea menor a tres, muestra un mensaje indicando que el número de parámetros es inválido.



1.3. Ejemplo 3: calculadora-web.js

Este ejemplo es una versión mejorada del anterior ya que, además de poder realizar operaciones escribiendo una url concreta, permite realizar operaciones de forma interactiva. Internamente, la parte interactiva se trata como si el usuario hubiera introducido una url a través del fichero calc.html que hace uso de un script para crear la url.

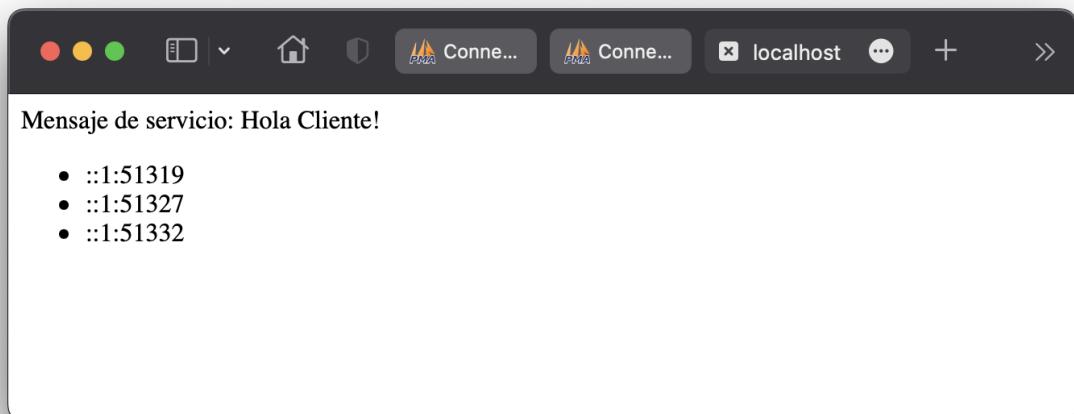




```
[mariolopg ejemplos (P4) $ node calculadora-web.js
Servicio HTTP iniciado
Petición REST: sumar/2/3
Petición REST: sumar/2/3
Petición REST: sumar/2/3
```

1.4. Ejemplo 4: connections.js

Este ejemplo permite comprobar los clientes conectados junto con el puerto asociado de forma simultánea. Cada vez que se conecta un cliente nuevo o se desconecta otro, el cambio se refleja de forma automática en el resto de clientes. En la terminal se muestran las conexiones y desconexiones junto con los puertos correspondientes.

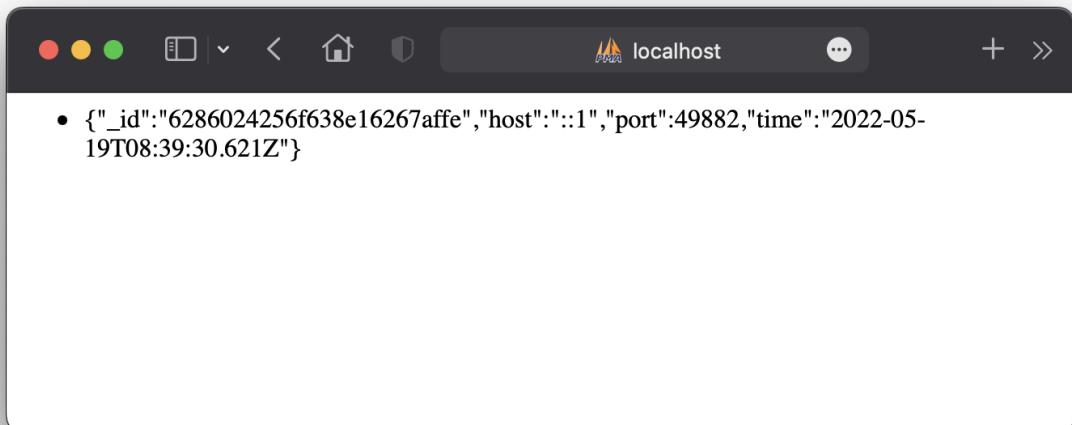


```
Mensaje de servicio: Hola Cliente!
● ::1:51319
● ::1:51327
● ::1:51332
```

```
[mariolopg ejemplos (P4) $ node connections.js
Servicio Socket.io iniciado
New connection from ::1:51319
New connection from ::1:51319
El cliente ::1 se va a desconectar
[ { address: '::1', port: 51319 }, { address: '::1', port: 51319 } ]
El usuario ::1 se ha desconectado
New connection from ::1:51327
New connection from ::1:51332
[]
```

1.5. Ejemplo 5: mongo-test.js

Este ejemplo proporciona la información sobre conexiones a la base de datos realizadas por los clientes junto con su id, puerto y hora exacta en la que han realizado la conexión.



```
[mariolopg ejemplos (P4) $ node mongo-test.js
Servicio MongoDB iniciado
(node:23917) [MONGODB DRIVER] Warning: collection.insert is deprecated. Use insertOne, insertMany or bulkWrite instead.
(Use `node --trace-warnings ...` to show where the warning was created)]
```

2. Practica 4: Node.JS - IOT Simulator

2.1. httpServer.js

Para la realización de este simulador he realizado en una primera instancia la clase httpServer, utilizada para crear un servidor http y servir las peticiones. El constructor admite como argumentos la base de datos y opcionalmente un puerto ya que por defecto escucha en el puerto 8080.

```
constructor (dataBase, port = 8080){
    this.port = port;
    this.dataBase = dataBase;
    this.server = http.createServer(
        ((request, response) => {
            var ruta = this.ruta(request.url);
            if(request.url == "/historico/lectura" || request.url == "/home/lectura" || request.url == "/controlador/lectura"){
                response.writeHead(200, mimeTypes["json"]);
                this.dataBase.getData("switch-aire").then((data) => {
                    this.socketio.emit("historico-aire", data)
                })
                this.dataBase.getData("switch-persiana").then((data) => {
                    this.socketio.emit("historico-persiana", data)
                })
                this.dataBase.getData("slider-temperatura").then((data) => {
                    this.socketio.emit("historico-temperatura", data)
                })
                this.dataBase.getData("slider-luminosidad").then((data) => {
                    this.socketio.emit("historico-luminosidad", data)
                })
                response.end();
            }
            else
                this.leerPagina(response, ruta)
        })
    );
    this.socketio = new SocketIOServer(this.server)
}
```

Si llega una petición de histórico envío un emit con el socket para que la página solicitante obtenga las últimas actualizaciones de la base de datos.

En él transformo la url para obtener el fichero html correspondiente de la siguiente forma:

```

ruta(uri){
    var ruta = "";
    var directorio = "public/pages/";
    switch (uri) {
        case "/": case "/home":
            ruta = directorio + "home.html";
            break;

        case "/controlador":
            ruta = directorio + "controlador.html";
            break;

        case "/historico":
            ruta = directorio + "historico.html";
            break;

        case "/404":
            ruta = directorio + "404.html";
            break;

        default:
            var peticion = uri.slice(1).split("/");
            if(peticion[0] == 'controlador' && peticion[1] != 'lectura'){
                ruta = directorio + "controlador.html";
                console.log(peticion[1] + " " + peticion[2])
                var evento = {time:getTimeStamp(), valor:peticion[2]};
                this.dataBase.insertar(peticion[1], evento)
                this.socketio.emit(peticion[1], evento)

                // Emision de alerta
                if(peticion.length == 4){
                    this.socketio.emit("alerta", peticion[3])
                }
            }
            else{
                ruta = path.join(process.cwd(), uri);
            }
            break;
    }

    return ruta;
}

```

El default es utilizado para las emisiones en caso de cambio en los sensores y, en caso de que la temperatura o luminosidad superen unos umbrales, emitir una alerta. Esto se realiza con un emit desde el socket.

Una vez obtengo el fichero html a partir de la url, a través de la función fs leo el fichero y lo envío.

```
leerPagina(response, fname){
    fs.readFile(fname, function (err, data) {
        if (!err) {
            var extension = fname.split(".")[1];
            var type = mimeTypes[extension];
            var code = 200;
            response.writeHead(code, type);
            response.write(data);
        }
        else {
            response.writeHead(301, { "Location": "/404" });
        }

        response.end();
    });
}
```

Para lanzar el servidor he creado un método que inicia el socket y pone al servidor a escuchar en el puerto establecido.

```
iniciar() {
    this.socketio.iniciar();
    this.server.listen(this.port);

    //Compruebo si el servidor se inicia o no.
    if (this.server.listening) {
        console.log("Servicio HTTP iniciado");
    } else {
        throw new Error("Error")
    }
}
```

2.2. dbServer.js

Esta clase es utilizada para la creación de la base de datos, crear, insertar filas y obtener la información de una tabla.

En el constructor le indico la url de la base de datos, puerto al que enviar información y nombre de la base de datos.

```
constructor(url = "mongodb://localhost:27017/", port = 8080, nombre = "DSDP4"){
    this.url = url;
    this.port = port;
    this.nombre = nombre;
    this.dataBase = null;
}
```

Al momento de iniciar el servicio el cliente Mongo se conecta a la base de datos, si no existe la crea,

y almacena una referencia a la misma.

```
iniciar(){
    MongoClient.connect(this.url, { useUnifiedTopology: true })
    .then( (connection) => {
        this.dataBase = connection.db(this.nombre);
        console.log("Servidor Mongo iniciado");
    })
    .catch((err) => {
        console.log("Error al iniciar servidor Mongo");
    })
}
```

Las operaciones que he permitido realizar son la búsqueda e inserción sobre tablas, como en el caso anterior, a la hora de insertar si no existe una tabla la crea.

```
async getData(nombre){
    var data = await this.dataBase.collection(nombre).find().toArray();
    return data;
}

async insertar(nombre, valores){
    return await this.dataBase.collection(nombre).insertOne(valores);
}
```

Las operaciones las declaro con `async` para que en caso de que se llame a la función dos o más veces al mismo tiempo se cree una cola y asegurar la exclusión mutua.

2.3. socketio.js

Esta clase se encarga de realizar las emisiones de mensajes hacia los sockets. En el constructor se pasa como argumento el servidor y se crea un socket asociado a dicho servidor.

```
var socketio = require("socket.io");

class SockerIOServer {
    constructor(httpServer){
        this.socket = socketio(httpServer);
    }

    emit (event, data) {
        this.socket.sockets.emit(event, data);
    }

    iniciar(){
        console.log("Socket IO iniciado");

        this.socket.sockets.on( 'connection', (socket) => {
            socket.on('start-session', (data) =>{
                var sessionID = data.sessionID;
                socket.emit('set-session-acknowledgement', {
                    sessionID,
                    name: data.name
                })
            })
        })
    }
}

module.exports = SockerIOServer;
```

2.4. index.js

Para lanzar todos los servicios he creado un archivo llamado index.

```
var HttpServer = require("./src/servidores/httpServer.js")
var DbServer = require("./src/servidores/dbServer.js");

var dbServer = new DbServer();
var httpServer = new HttpServer(dbServer);

dbServer.iniciar();
httpServer.iniciar();
```

Para iniciar la aplicación hay que realizar el siguiente comando:

```
$ node index.js
```

2.5. Notificación del cambio de sensores y/o dispositivos

Para la notificación del cambio en los sensores realizo un get con AJAX cada vez que realizo un cambio para que el servidor reciba una petición y la procese enviando las emisiones correspondientes.

```
var HttpServer = require("./src/servidores/httpServer.js")
var DbServer = require("./src/servidores/dbServer.js");

var dbServer = new DbServer();
var httpServer = new HttpServer(dbServer);

dbServer.iniciar();
httpServer.iniciar();
```

Para la recepción de eventos uso la función on de la clase socket y según el fichero donde se reciba

cambia unos elementos u otros.

Ejemplo del botón del aire en el fichero controlador.js

```
socket.on("switch-aire", function (data) {
    var status = false;
    if(data.valor == "encendido")
        status = true;
    setToggleStatus("switch-aire", status)
})
```

Para las alertas hay otra función como la anterior que realiza un alert del mensaje enviado. Ejemplo del botón del aire en el fichero controlador.js

```
socket.on("alerta", function (data) {
    var alerta = parseMsg(data);
    alert(alerta)
})

function parseMsg(data) {
    var pieces = data.split(' ');
    var msg = "";
    pieces.forEach(element => {
        msg += element + " ";
    });
    return msg;
}
```

2.6. Ejemplos de ejecución

La primera figura muestra la aplicación al iniciarla en tres dispositivos distintos.

The image shows three separate browser windows side-by-side, all displaying the "IOT Simulator System" interface at localhost:8080.

- Información Tab:** Shows device status. "Aire acondicionado" is labeled as "apagado" (off) in red. "Persiana" is also labeled as "apagado" (off) in red.
- Controlador Tab:** Shows controls for devices. Under "Dispositivos", there are two buttons: "Aire acondicionado" (which is now "encendido" - on, indicated by a blue circle) and "Persiana". Under "Sensores", there are sliders for "Temperatura" (set to 35°C) and "Luminosidad" (set to 50lm).
- Histórico Tab:** Shows historical data. It lists the state changes for each device: "Aire acondicionado" was turned on at 24/05/2022 - 11:23:41, "Persiana" was turned off, and both "Temperatura" and "Luminosidad" were recorded at their current values.

En este ejemplo enciendo el dispositivo de aire acondicionado y se actualizan los demás dispositivos.

This set of screenshots shows the same three browser tabs after the user has turned on the air conditioner via the Controlador tab.

- Información Tab:** Now shows "Aire acondicionado" as "encendido" (on) in blue, and "Persiana" as "apagado" (off) in red.
- Controlador Tab:** The "Aire acondicionado" button is now blue ("encendido"). The "Persiana" button is gray ("apagado"). The "Sensores" sliders remain at their previous positions (35°C and 50lm).
- Histórico Tab:** The historical log now includes the event: "24/05/2022 - 11:23:41 -> se ha encendido" (the air conditioner has been turned on).

En este ejemplo bajo la luminosidad en el sensor y lanza una alerta a los demás dispositivos indicando que la persiana se ha subido porque había poca luz.

The image displays three separate browser windows side-by-side, all showing the "IOT Simulator System" interface at localhost:8080.

- Left Window (Información):** Shows the "Estado de los dispositivos" section with "Aire acondicionado: encendido" and "Persiana: encendido". It also shows the "Estado de los sensores" section with "Temperatura: 35°C" and "Luminosidad: 5".
- Middle Window (Controlador):** Shows the "Dispositivos" section with "Aire acondicionado" (on) and "Persiana" (off). It shows the "Sensores" section with "Temperatura: 35°C" and "Luminosidad: 5lm". A modal dialog box is open, displaying the message "localhost:8080 dice" and "Se ha subido la persiana porque entra muy poca luz", with a blue "Aceptar" button.
- Right Window (Histórico):** Shows a list of historical events:
 - Aire acondicionado:** 24/05/2022 - 11:23:41 -> se ha encendido
 - Persiana:** 24/05/2022 - 11:24:09 -> se ha encendido
 - Temperatura:** 24/05/2022 - 11:24:09 -> Lum cambia a 5
 - Luminosidad:** 24/05/2022 - 11:23:56 -> Lum cambia a 96