

Documentación Técnica - AirHealthy

Modelo Fijo

Sistema de Monitoreo de Niveles de Dióxido de Carbono (CO₂) presentes en el aire usando ESP32 y un Raspberry PI



Índice

Introducción	4
Dependencias del Servidor	5
NodeJS	5
NPM	5
Bibliotecas	5
Variables Globales	7
Modelo de Red WLAN	8
Datos de Conexión para Interfaz de red 'Interna' del Raspberry Pi	9
Detalles del Servidor	10
1. Propósito del Servidor	10
2. Configuración y Conexión	10
3. Puntos de Conexión (endpoints)	11
4. Métodos de Autenticación	12
Diccionario de Datos - SQLite	13
Componentes del Sistema	14
Conexión al Servidor por medio de SSH	16
Requerimientos H&W	17
Funciones Esenciales del Servidor	18
1. Función: serverEnd()	18
2. Función: serverListen()	18
3. Función: exception()	19
Funciones del Código Fuente	20
1. __POST__REGISTRAR_MEDICION__()	20
Descripción	20
Parámetros	20
Proceso	20
Errores y Excepciones	23
2. __GET__MEDICIONES_PROMEDIADO__()	24
Descripción	24
Parámetros	24
Proceso	24
Errores y Excepciones	26
3. __GET__LISTA_MEDICIONES__()	27
Descripción	27
Parámetros	27
Proceso	27
Errores y Excepciones	28
4. __GET__DASHBOARD__()	29
Descripción	29
Parámetros	29
Proceso	29
Notas Adicionales	30
Errores y Excepciones	30
5. __GET__DESCARGAR__()	31
Descripción	31
Parámetros	31

Proceso	31
Errores y Excepciones	32
Funciones Complementarias	33
1. Función: obtenerTablaArp(dev)	33
Argumentos	33
Salida	34
Errores	34
2. Función: obtenerNombreHost(ip)	35
Argumentos	35
Salida	35
Errores	35
3. Función: lista_arp(dev)	36
Argumentos	36
Salida	36
Errores	36
4. Función: lista_medidores()	37
Argumentos	37
Salida	37
Errores	37
5. Exportación de las Funciones	38
Se exportan todas las funciones que se harán uso en el archivo socket.js del proyecto (Imagen 9.8).	38

Introducción

En la era actual, donde la preocupación por la calidad del aire y el medio ambiente está en constante aumento, se vuelve crucial desarrollar tecnologías que nos permitan monitorear y comprender mejor nuestra atmósfera. En este contexto, surge nuestro proyecto de un sensor de calidad de aire utilizando componentes como el sensor de gas MQ-135 y el sensor de temperatura y humedad DHT-11, integrados con el poderoso microcontrolador ESP32. El objetivo principal de este proyecto es proporcionar una solución innovadora y accesible para la medición y monitoreo en tiempo real de la calidad del aire.

Para lograr este propósito, hemos diseñado un sistema completo que no solo recopila datos de manera precisa y confiable, sino que también los transmite de forma eficiente a un servidor centralizado. Este servidor, alojado en una Raspberry Pi, utiliza tecnologías como Node.js y SQLite para gestionar la recepción, almacenamiento y procesamiento de los datos ambientales proporcionados por los sensores.

Una de las características destacadas de nuestro sistema es la interfaz web que proporciona una visualización clara y detallada de los datos recopilados. Esta interfaz permite a los usuarios monitorear en tiempo real los niveles de dióxido de carbono, temperatura y humedad del ambiente, además de ofrecer estadísticas útiles como el promedio de datos de la última hora, del día actual y de las últimas 24 horas. Además, hemos implementado una funcionalidad que permite la descarga de los datos almacenados en un formato conveniente, facilitando su análisis y uso posterior.

Dependencias del Servidor

NodeJS

Node.js es un entorno de ejecución de JavaScript en el lado del servidor. Utiliza el motor V8 de Google, es conocido por su manejo asíncrono eficiente, y viene con NPM para gestionar dependencias. Ampliamente utilizado en el desarrollo web, es versátil, de código abierto y permite a los desarrolladores utilizar JavaScript para construir aplicaciones del lado del servidor (Imagen 1.1).



Imagen 1.1. Entorno de Ejecución NodeJS

NPM

NPM, o Node Package Manager, es un componente fundamental del ecosistema de Node.js que facilita la gestión de dependencias en proyectos de software. Permite a los desarrolladores instalar, compartir y gestionar bibliotecas y herramientas utilizando comandos simples como `npm install`. Además, sirve como un repositorio global que alberga una amplia variedad de paquetes públicos que pueden ser incorporados en proyectos. Con la capacidad de automatizar tareas mediante scripts y seguir un sistema de versionado semántico, NPM mejora la eficiencia en el desarrollo de software en Node.js (Imagen 1.2).



Imagen 1.2. Gestor de Paquetes NPM

Bibliotecas

La interfaz de programación (API) requiere de algunas dependencias para su correcto funcionamiento. No olvidemos que usamos NodeJS para darle forma al servidor.

En el caso del archivo **constantes.js**, se hace uso de las bibliotecas correspondientes (Imagen 1.3).

```
const net = require('net');
const util = require('util');
const arp = require('arp-a');
const dns = require('dns');
```

Imagen 1.3. Bibliotecas del archivo "constantes.js" perteneciente al proyecto del servidor.

Por el otro lado, tenemos al archivo con nombre **socket.js** el cual dispone también de las bibliotecas necesarias para hacer posible el servidor (Imagen 1.4).

```
const http = require('http');
const mUrl = require('url');
const IP = require('ip');
const fs = require('fs');
const requestIP = require('request-ip');
const sqlite3 = require('sqlite3');
const { isValid, parseISO } = require('date-fns');
```

Imagen 1.4. Bibliotecas del archivo “socket.js” perteneciente al proyecto del servidor.

Y por último **constantes** y **server**, los cuales no son bibliotecas de Node.js, más bien son módulos (archivos del proyecto) que existen en el directorio de nuestro sistema, es decir, del servidor y estos contienen las funciones complementarias y esenciales para hacer funcionar el servicio (Imagen 1.5).

```
const funciones = require('./constantes');
const _server = require('./server');
```

Imagen 1.5. Importación de archivos “constantes.js” y “server.js” pertenecientes al directorio del proyecto.

Variables Globales

Las variables globales en el código cumplen funciones clave en la configuración y funcionamiento del servidor, así como en la gestión de datos. Establecen la configuración del servidor, como el puerto en el que escucha las solicitudes y la ruta a la base de datos. También facilitan el acceso y la manipulación de datos, permitiendo la conexión con la base de datos y proporcionando un nombre para la tabla donde se almacenan las mediciones. Estas variables son esenciales para garantizar que el servidor funcione correctamente y que los datos se gestionen de manera eficaz (Imagen 2.1 e Imagen 2.2).

A continuación se presenta la manera en que se declaran las variables en el servidor.

```
const puerto = 4040;
var server;

const dbroute = './db/airhealthy.db';
const table = 'mediciones';
var db;
```

Imagen 2.1.

```
// Definir el objeto para manejar las autenticaciones del sistema
const authenticationHashes = {
  'GET': {
    '/mediciones': 'c8cfdd77ac4061a8a56f84614c23942c',
    '/mediciones/prom': '312265b9ed1e8a46e94f289921e0db4d',
    '/mediciones/prom/hoy': '2cf6f5b98a7d7b266e80d663d8756852',
    '/mediciones/prom/dia': 'dd459a4e71abe042aa74a0c34614f7d0',
    '/mediciones/prom/hora': '98c41205d51b6a99d2c5992bfa767a9a'
  },
  'POST': {
    '/medicion': '9dae27a1e26d7445e30ec3fa4b722667'
  }
};
```

Imagen 2.2

Modelo de Red WLAN

1. Cliente (sensor) - Servidor

Referencia de Imagen: [Imagen 3.1]

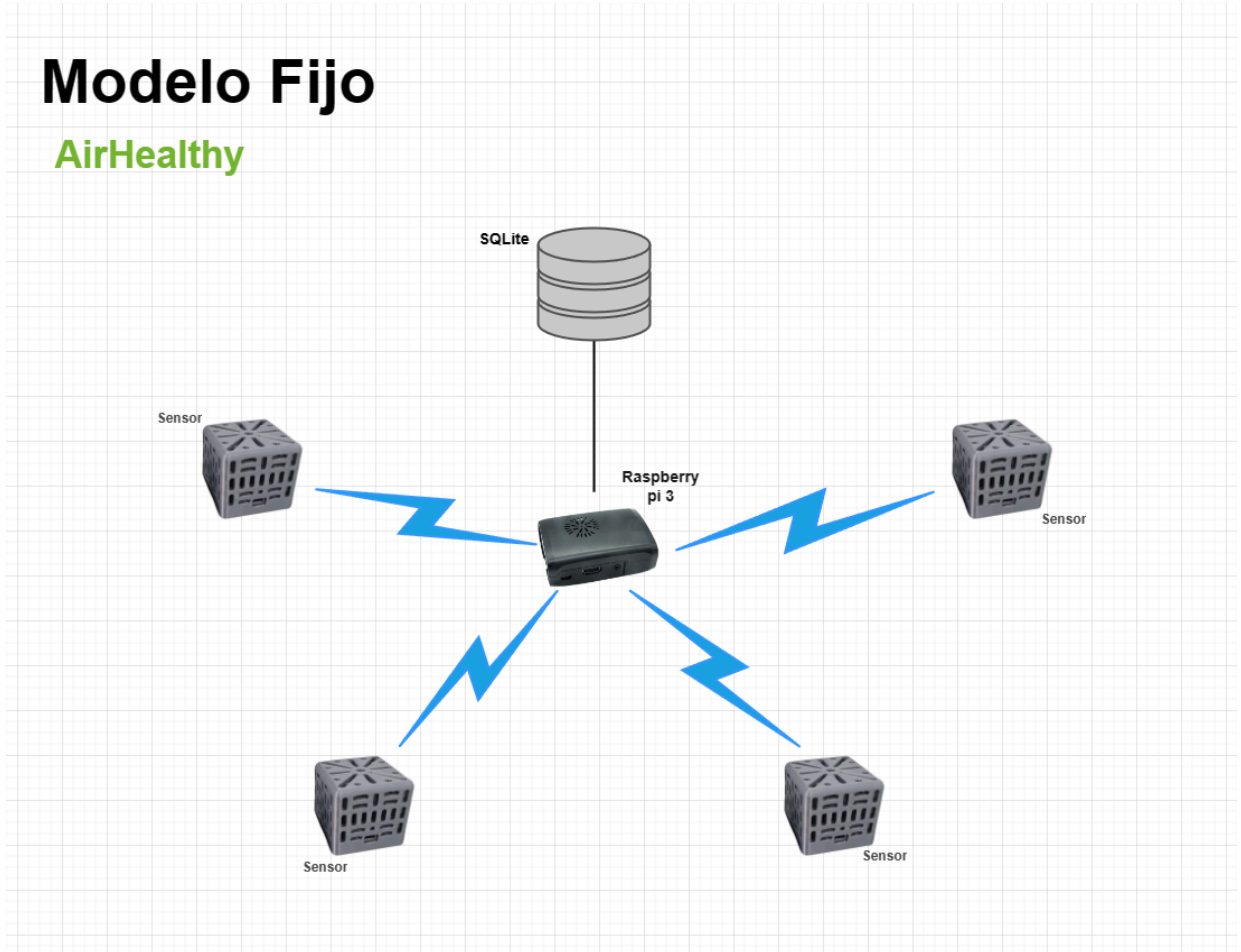


Imagen 3.1. Modelo de red WLAN del servidor en función de los sensores.

2. Cliente (persona) - Servidor

Referencia de Imagen: [Imagen 3.2]

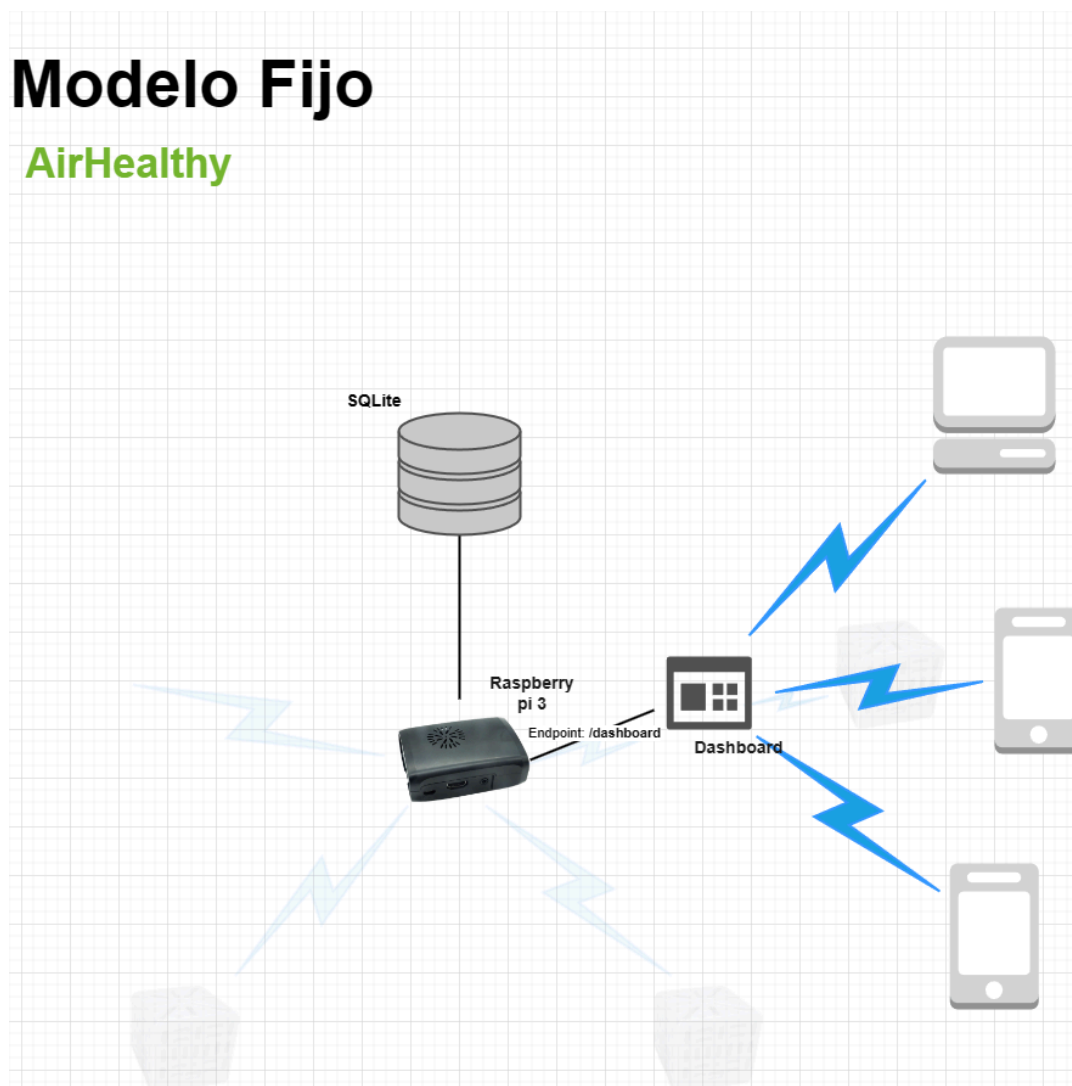


Imagen 3.2. Modelo de red WLAN del servidor en función de los clientes usando dispositivos personales.

Datos de Conexión para Interfaz de red 'Interna' del Raspberry Pi

☒ **SSID:** airasphealthy

☒ **WPA-PSK:** raspados1108

☒ **TIPO:** WPA2

Detalles del Servidor

1. Propósito del Servidor

Partiendo del concepto de IoT, este servidor es una Interfaz de Programación de Aplicaciones (API) que permite la comunicación entre los ESP32 y la Raspberry Pi. El propósito de este servicio es que los Módulos de Sensores Fijos (ESP32) envíen los datos correspondientes al servidor que se encargará de almacenar y procesar los datos para devolver la información solicitada por el lado del cliente (persona) visualizándolo finalmente desde una página web.

2. Configuración y Conexión

- **Dirección del Host:** 192.168.10.1
- **Puerto:** 4040
- **Protocolo:** HTTP
- **URL Base:** <http://192.168.10.1:4040/>

3. Puntos de Conexión (endpoints)

Endpoints	Método HTTP	Invoca la función	Parámetros de la Función
/mediciones	GET	GET_LISTA_DE_MEDICIONES()	1. request 2. response
/mediciones/prom	GET	GET_MEDICIONES_PROMEDIADO()	1. request 2. response 3. date (<i>string</i>) a. "xdia" b. "hoy" c. "xhora" 4. value (<i>date time</i>)
/mediciones/prom/hoy	GET		
/medicion	POST	POST_REGISTRAR_MEDICION()	1. request 2. response 3. json (<i>object</i>) a. co2 b. temp c. hum
/dashboard	GET	GET_DASHBOARD()	1. request 2. response
/descargar	GET	GET_DESCARGAR()	1. request 2. response

4. Métodos de Autenticación

El mecanismo de autenticación se basa en un método básico que funciona mediante validación de credenciales (Imagen 4.1).

En este caso, se cuenta con un objeto (JSON) simple que almacena la credencial correspondiente para cada punto de conexión. El objeto cuenta con la siguiente estructura:

```
const authenticationHashes = {
  'GET': {
    '/mediciones': '<hashForGetEndpoint1>',
    '/mediciones/prom': '<hashForGetEndpoint2>',
    '/mediciones/prom/hoy': '<hashForGetEndpoint3>',
    '/mediciones/prom/dia': '<hashForGetEndpoint4>',
    '/mediciones/prom/hora': '<hashForGetEndpoint5>'
  },
  'POST': {
    '/medicion': '<hashForPostEndpoint1>'
  }
};
```

Imagen 4.1. Variable Global que permite almacenar los hash (claves encriptadas) que requiere el cliente (sensor) para enviar datos al servidor y visualizar la información de los promedios si se requiere.

De esta manera, nos aseguramos que tras cada cliente (sea sensor o una persona) que intente lanzar una solicitud al servidor, disponga del hash o credencial correspondiente que le permita identificarse y consumir del recurso que sea necesario (Imagen 4.2).

```
const providedHash = (request.headers.authorization || '').split(' ')[1] || '';

// Obtiene el hash válido para la combinación de método y endpoint
const validHash = authenticationHashes[method][parsedUrl.pathname];

if (providedHash !== validHash) {
  response.writeHead(401, { 'WWW-Authenticate': 'Basic realm="Secure Area"' });
  const respuesta = {
    success: false,
    status: 401,
    response: true,
    message: "Acceso Denegado. No dispones de las credenciales adecuadas."
  };

  response.end( JSON.stringify( respuesta ) );
  return;
}
```

Imagen 4.2. Proceso (algoritmo) del servidor que permite validar las claves encriptadas enviadas por el cliente al momento de solicitar una petición al mismo.

Diccionario de Datos - SQLite

Nombre de la Tabla	mediciones
Descripción de la Tabla	<p>Esta tabla permite almacenar 4 datos esenciales + índice primario (primary key) de la tabla.</p> <p>Cuyos datos son los siguientes:</p> <ol style="list-style-type: none">1. Co22. Temperatura3. Humedad4. FechaHora
Columnas de la Tabla	<ul style="list-style-type: none"><input checked="" type="checkbox"/> Id<input checked="" type="checkbox"/> Co2<input checked="" type="checkbox"/> Temperatura<input checked="" type="checkbox"/> Humedad<input checked="" type="checkbox"/> FechaHora
Relaciones	
Índices	<ul style="list-style-type: none">• Id (Primary Key)
Extras	<ul style="list-style-type: none">• Columna Id → Auto Incrementable
Procedimientos Almacenados	
Triggers	

Componentes del Sistema

Raspberry PI	ESP32
<ul style="list-style-type: none">● Responsabilidad: Servidor central.● Sistema Operativo: Raspbian OS● Tecnologías Implementadas: Node.js y SQLite.● Conexiones: Recibe datos de los ESP32 por medio de HTTP.● Interfaz de Red:<ul style="list-style-type: none">☑ Nombre → “Interna”☑ SSID → airasphealthy☑ WPA-PSK → raspados1108☑ Dirección IPv4 → 192.168.10.1☑ Rango de IP’s → 192.168.10.#☑ Máscara de Red → 255.255.255.0 (192.168.10.0 - 192.168.10.255)	<ul style="list-style-type: none">● Responsabilidad: Recoger y enviar datos de monitoreo.● Sensores: CO2 (MQ-135), Temperatura y Humedad (DHT-11).● Protocolo de Comunicación: HTTP.● Conexiones: Se comunica con la Raspberry Pi mediante WLAN.
Servidor (Node.js)	
<ul style="list-style-type: none">● Responsabilidad: Procesar y almacenar datos recibidos.● Tecnologías: Express.js, SQLite3 driver.● API Endpoints:<ul style="list-style-type: none">☑ <i>/mediciones</i>☑ <i>/mediciones/prom</i>☑ <i>/mediciones/prom/hoy</i>☑ <i>/medicion</i>☑ <i>/dashboard</i>Consultar más detalles.	

Base de Datos

- **Nombre o tipo:** SQLite
- **Responsabilidad:** Almacenamiento persistente de los datos recolectados por los sensores.
- **Tablas:**
 1. datos_sensores
- **Relaciones:** Ninguna.
- **Índices:**

Tipo	Campo	Tabla
Primary Key	Id	datos_sensores

Conexión al Servidor por medio de SSH

Para realizar una conexión al servidor usando ssh debemos realizar los siguientes pasos:

1. Abrir la ventana de línea de comandos de su sistema operativo.
2. Escribir lo siguiente: `ssh <nombreusuario>@direccion_ip`
3. Reemplazar nombreusuario por “**cynan**” (sin comillas) y en direccion_ip por “**192.168.10.1**” (sin comillas).

Nota: asegurarse de estar conectado a la red airaspehealthy antes de proceder con la conexión, de lo contrario no se podrá establecer comunicación.

4. Al presionar enter, nos pedirá la contraseña de acceso al servidor. Para este caso ingresamos la siguiente: “**pi**” (sin comillas).

Con estos pasos, habremos podido ingresar al servidor.

Requerimientos H&W

- **Requerimientos de Software**

- Sistema Operativo: GNU/Linux
- Distribución: Debian o Raspberry Pi OS (Raspbian)
- Paquetes: *hostapd*, *dnsmasq*, *iptables*.
- NodeJS (\geq v18.x)

- **Requerimientos Mínimos de Hardware**

- CPU: Quad-Core, Single-Thread a 1.2GHz
- RAM: 1GB LPDDR2 a 900MHz
- Conectividad: Ethernet, WiFi 802.11 b/g/n (2.4 GHz)
- Almacenamiento: MicroSD 32GB (Clase 10)

Funciones Esenciales del Servidor

server.js

Esté módulo consiste de tres funciones principales, que nos permiten manejar eventos y respuestas del servidor.

1. Función: serverEnd()

- **Objetivo:** Esta función maneja los eventos cuando un cliente se desconecta del servidor.
- **Parámetros:**
 - **req:** Objeto de solicitud HTTP entrante.
 - **res:** Objeto de respuesta HTTP saliente.
- **Proceso:**
 - Genera y registra una marca de tiempo que indica cuándo un cliente se ha desconectado (Imagen 7.1).
- **Código Relevante:**

```
const date = new Date();  
const datetime = date.toLocaleString().replace(/,/g, '');  
console.log("> Cliente desconectado (", datetime, ").\n");
```

Imagen 7.1. Proceso de la función serverEnd()

2. Función: serverListen()

- **Objetivo:** Esta función notifica que el servidor está escuchando en un puerto específico.
- **Parámetros:**
 - **p:** es un número (entero) que especifica en donde está escuchando el servidor.
- **Proceso:**
 - Imprime un mensaje en la consola indicando el puerto en el que el servidor está escuchando y a su vez sirve para informar que el servidor se encuentra activo y funcionando correctamente (Imagen 7.2).
- **Código Relevante:**

```
console.log("> Servidor escuchando en el puerto: ", p);
```

Imagen 7.2. Proceso adicional de la función que imprime mensaje cuando el servidor se encuentra en línea.

3. Función: exception()

- **Objetivo:** Esta función maneja las excepciones, generando una respuesta JSON con un estado 404 cuando no se puede generar ninguna otra respuesta.
- **Parámetros:**
 - **req:** Objeto de solicitud HTTP entrante.
 - **res:** Objeto de respuesta HTTP saliente.
- **Proceso:**
 - Crea un objeto con un estado de 404 y un mensaje de error.
 - Escribe el código de estado 404 en la respuesta HTTP.
 - Envía el objeto de error como una cadena JSON en la respuesta HTTP.

Referencia de Imagen: [Imagen 7.3]

- **Código Relevante:**

```
const objeto = {  
  status: 404,  
  message: 'No se generó ninguna respuesta.',  
  response: false  
};  
res.writeHead(404);  
res.end(JSON.stringify(objeto));
```

Imagen 7.3. Proceso de la función exception() que permite emitir una respuesta visual al cliente.

Funciones del Código Fuente

socket.js

1. __POST__REGISTRAR_MEDICION__()

Método HTTP: **POST**

Descripción	Parámetros
La función es un controlador del servidor Node.js destinado a procesar y registrar mediciones provenientes de solicitudes HTTP POST. La función valida los datos recibidos, los registra en una base de datos SQLite, y responde al cliente con detalles sobre el resultado del proceso.	<ul style="list-style-type: none">• req → Objeto que contiene información sobre la solicitud HTTP entrante, tales como los parámetros de la URL y los encabezados.• res → Objeto utilizado para enviar la respuesta HTTP al cliente.• json → Objeto que contiene el cuerpo de la solicitud HTTP en formato JSON, el cual incluye los datos de medición a registrar en la base de datos.

Proceso

- **Validación de Datos** → Verifica la presencia y validez del dato *co2*, respondiendo con un error 400 Bad Request si falta o no es un número.
- **Extracción y Conversión de Datos** → Extrae y convierte los datos *co2*, *temp*, y *hum* de json a enteros. Asigna 0 a *temp* y *hum* si no son números válidos.
- **Obtención de Dispositivos** → Obtiene una lista de dispositivos ESP32 conectados a la red interna del Raspberry Pi.
- **Identificación de Dirección IP** → Identifica y normaliza la dirección IP del dispositivo que realiza la solicitud.
- **Identificación de Dispositivo** → Busca el dispositivo en la lista de dispositivos por su dirección IP y valida si es un ESP32 reconocido.
- **Registro en la Base de Datos** → Si el dispositivo es un ESP32 válido, registra los datos recibidos en una base de datos SQLite, maneja los errores de inserción y cierra la conexión con la base de datos.
 - **Inicialización de la Base de Datos:** Se inicia la conexión con la base de datos de 'SQLite' usando el valor de la variable '**dbroute**' (Imagen 8.1).

```
const db = new sqlite3.Database(dbroute);
```

Imagen 8.1. Inicialización de la base de datos para preparar la lectura o inserción de datos.

- **Creación de Fecha y Hora:** Se obtiene la fecha y hora actuales en formato ISO 8601 (Imagen 8.2).

```
const date = new Date();  
const fh = date.toISOString();
```

Imagen 8.2. Método de creación de las fechas con el formato ISO (YYYY-MM-DD)

- **Inserción de Datos:** Se ejecuta una inserción en la tabla *mediciones* de la base de datos *airhealthy.db* (Imagen 8.3, Imagen 8.4).

```
db.run(`INSERT INTO ${table} (co2, temp, hum, ipAddress, macAddress, fechaHora) VALUES (?, ?, ?, ?, ?, ?)`,
```

Imagen 8.3. Preparación de la sintaxis SQL para la inserción de valores en una tabla.

```
, [nivelCo2, nivelTemperatura, nivelHumedad, ipv4, currentDevice.mac, fh]);
```

Imagen 8.4. Se proporcionan los datos que se almacenarán en la base de datos.

- **Cierre de la Base de Datos**

```
db.close();
```

Imagen 8.5. Función que permite cerrar la base de datos.

- **Respuesta al Cliente (Imagen 8.5)**

- **Éxito**
 - **Status Code** → 200 OK
 - **Cuerpo:**

```
{  
  "status": 200,  
  "message": "ok.",  
  "response": true  
}
```

Imagen 8.5. Datos que se emiten al cliente desde el servidor.

- **Dispositivo No Reconocido** (Imagen 8.6)

- **Status Code** → 200 OK

- **Cuerpo:**

```
{  
  "status": 200,  
  "message": "Advertencia. El dispositivo no es un microcontrolador ESP32",  
  "response": true  
}
```

Imagen 8.6. Datos que se emiten al cliente desde el servidor cuando un dispositivo no es un ESP32.

- **Error de Validación y Otros Errores** (Imagen 8.7)

- **Status Code** → 400 Bad Request
- **Cuerpo:**

```
{  
  "status": 400,  
  "message": "<Emitido por la Interfaz>",  
  "response": false  
}
```

Imagen 8.7. Datos que se emiten al cliente cuando ocurre un error de validación u otros errores afines.

Errores y Excepciones

- Los errores durante la ejecución, incluidos los datos inválidos y los problemas de base de datos, se capturan y se manejan mediante un bloque try/catch, y se envían al cliente en una respuesta HTTP con estado 400 Bad Request y un mensaje de error detallado.

2. __GET__MEDICIONES_PROMEDIADO__()

Método HTTP: **GET**

Descripción	Parámetros
<p>La función está diseñada para manejar solicitudes HTTP GET en un servidor Node.js, y tiene como objetivo recuperar y devolver promedios de mediciones (CO2, temperatura, y humedad) de una base de datos SQLite, pudiendo filtrar dichos promedios por día, hora, o para el día actual.</p>	<ul style="list-style-type: none">• req → Objeto que contiene información sobre la solicitud HTTP entrante, tales como los parámetros de la URL y los encabezados.• res → Objeto utilizado para enviar la respuesta HTTP al cliente.• date (opcional) → Cadena de texto que determina el tipo de filtrado por fecha:<ul style="list-style-type: none">○ 'xdia'○ 'hoy'○ 'xhora'• value (Opcional) → Objeto que contiene valores específicos que funcionan en conjunto con date para llevar a cabo el proceso de filtrado (puede incluir fecha o hora).

Proceso

- **Inicialización del Objeto de Respuesta al Cliente** (Imagen 8.8)

```
const info = {  
  status: 200,  
  message: 'ok',  
  response: true,  
  data: []  
};
```

Imagen 8.8. Variable con nombre "info" que funge como objeto con formato JSON que es emitido al cliente cuando el proceso de obtener el promedio de mediciones es exitoso.

- **Consulta a la Base de Datos**
 - **Conexión a la Base de Datos** (Imagen 8.1)
 - **Construcción de la Consulta SQL**

- Se comienza a elaborar la consulta SQL para recuperar el promedio de los datos de las mediciones.
- El filtrado de los resultados se realiza de acuerdo con los parámetros **date** y **value** recibidos.
- Si se proporcionan inválidos o no se cumplen con las condiciones de la función para el filtrado, se retorna como respuesta al cliente **'400'** y termina la ejecución.

- **Cierre de la Base de Datos** (Imagen 8.5)

- **Ejecución de la Consulta y Respuesta**

- La consulta construida se ejecuta y, en caso de éxito, los resultados se asignan a **info.data** y se envía el objeto **info** como respuesta con un status de **'200'**.
- En caso de error durante la consulta, se registra el error y se envía el objeto **info** con un status de **'400'**.

- **Respuestas al Cliente** (Imagen 8.9)

- a) Éxito:

- **Status Code** → 200 OK.
- **Cuerpo:**

```
{
  "status": 200,
  "message": "ok",
  "response": true,
  "data": [] // Aquí se reciben el promedio de los datos sensados por
}
```

Imagen 8.9. Objeto con formato JSON emitido al cliente cuando se ha procesado exitosamente el promedio de los datos y a su vez se ha podido validar la clave encriptada.

- b) Error de Base de Datos o Parámetros Inválidos (Imagen 8.10):

- **Status Code** → 400 Bad Request.
- **Cuerpo:**

```
{
  "status": 400,
  "message": "disapproved",
  "response": true,
  "data": [] // No se recibe ningún dato
}
```

Imagen 8.10. Objeto con formato JSON que es emitido al cliente cuando ha ocurrido un error al procesar la solicitud, ya sea por clave encriptada errónea o por consulta a la base de datos.

Errores y Excepciones

- En caso de error durante la consulta, se registra el error, se modifica el objeto info con un status de 400, y se envía como respuesta (Imagen 8.10).
- Si la consulta es exitosa, los datos recuperados se asignan a info.data y se envía el objeto info como respuesta con un status de 200 (Imagen 8.9).

3. __GET__LISTA_MEDICIONES__()

Método HTTP: **GET**

Descripción	Parámetros
La función está diseñada para manejar solicitudes HTTP GET en un entorno de servidor Node.js, su principal tarea es recuperar y devolver una lista de mediciones almacenadas en una base de datos SQLite. La función responde con un objeto JSON que contiene el estado del proceso, un mensaje descriptivo, y los datos recuperados de la base de datos en caso de éxito.	<ul style="list-style-type: none">• req → Objeto que contiene información sobre la solicitud HTTP entrante, tales como los parámetros de la URL y los encabezados.• res → Objeto utilizado para enviar la respuesta HTTP al cliente.

Proceso

- **Inicialización del Objeto de Respuesta al Cliente** (Imagen 8.8)
- **Consulta a la Base de Datos**
 - **Conexión a la Base de Datos** (Imagen 8.1)
 - **Obtener registro de tabla 'mediciones'** (Imagen 8.11)

En esta parte se realiza una consulta SQL a la base de datos (airhealthy) para recuperar todas las filas de la tabla 'mediciones' y así posteriormente entregarlas como respuesta al cliente. Adicionalmente se extrae la fecha y hora (por separado) del registro correspondiente.

```
const camposAdicionales = 'DATE(fechaHora) AS func_date_fechaHora, TIME(fechaHora) AS func_time_fechaHora';
db.all(`SELECT *, ${camposAdicionales} FROM ${table} ORDER BY id DESC`,[], function(err,rows) {...});
```

Imagen 8.11. Proceso de consulta a la tabla mediciones de la base de datos.junto con campos adicionales de fecha y la hora.

- **Cierre de la Base de Datos** (Imagen 8.5)
- **Respuesta al Cliente**
 - a) **Éxito:**
 - **Status Code** → 200 OK.
 - **Cuerpo:**
Referencia de Imagen: [Imagen 8.9]
 - b) **Error de Base de Datos:**
 - **Status Code** → 400 Bad Request.
 - **Cuerpo:**
Referencia de Imagen: [Imagen 8.10]

Errores y Excepciones

- En caso de error durante la consulta, se registra el error, se modifica el objeto info con un status de 400, y se envía como respuesta.
- Si la consulta es exitosa, los datos recuperados se asignan a info.data y se envía el objeto info como respuesta con un status de 200.

4. __GET__DASHBOARD__()

Método HTTP: **GET**

Descripción	Parámetros
La función está diseñada para el cliente y maneja solicitudes HTTP GET en un entorno de servidor Node.js, su principal tarea es mostrar una página html, el cual procesa y describe todos los datos recuperados de los sensores y promedios de los datos bajo ciertas condiciones (todos, ultima hora, hoy y de las últimas 24 horas).	<ul style="list-style-type: none">• req → Objeto que contiene información sobre la solicitud HTTP entrante, tales como los parámetros de la URL y los encabezados.• res → Objeto utilizado para enviar la respuesta HTTP al cliente.

Proceso

- Se construye un html usando estilos de bootstrap (Imagen 8.12)

```
const bootstrapMinCss = `
```

- **Respuesta al Cliente (Imagen 8.14)**

a) **En caso de respuesta “Éxito”:**

- **Se muestra el siguiente contenido en pantalla:**

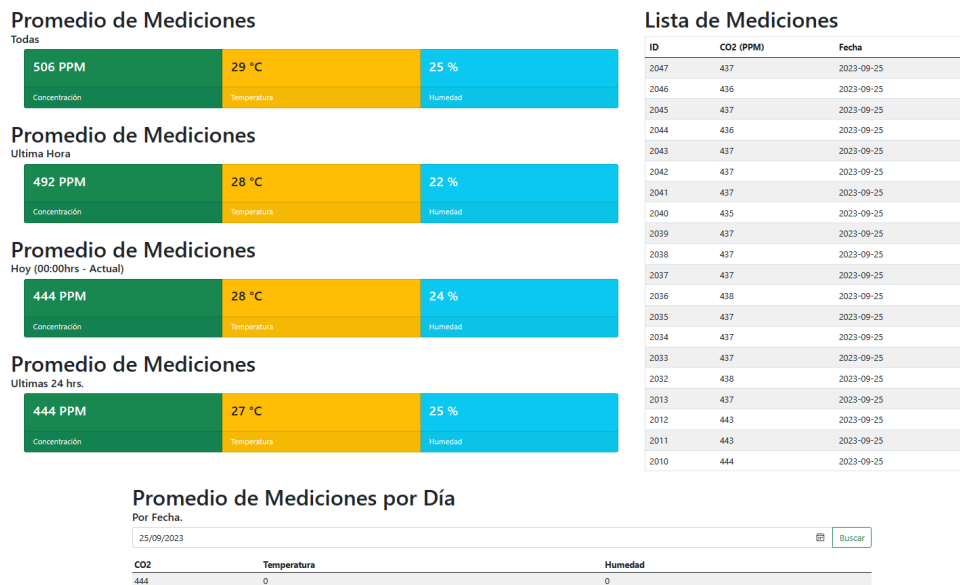


Imagen 8.14. Respuesta emitida al cliente, se visualiza una página html con datos de las mediciones.

b) **Error:**

En caso de algún error se informa en la consola del sistema y el cliente se desconecta de la sesión.

Notas Adicionales

- **Estilos Adicionales:**

- Para pantallas con un ancho mínimo de 768px, el ancho de la sección "Medidas Promedio por Día" es del 60%.
- Para pantallas con un ancho máximo de 768px, el ancho de la misma sección es del 100%.

Errores y Excepciones

- Si ocurre algún error durante la ejecución de la función, se captura y se registra en la consola con un mensaje descriptivo.

5. __GET__DESCARGAR__()

Método HTTP: **GET**

Descripción	Parámetros
La función está diseñada para el cliente y permite que el servidor realice una consulta sql a la tabla de <i>mediciones</i> , en los cuales todos los registros son recorridos usando la función map() y join() con el propósito de escribir la información cuyos datos están separados por comas y cada registro por saltos de líneas en un archivo de texto. Finalmente se emite al cliente el archivo generando su descarga.	<ul style="list-style-type: none">• req → Objeto que contiene información sobre la solicitud HTTP entrante, tales como los parámetros de la URL y los encabezados.• res → Objeto utilizado para enviar la respuesta HTTP al cliente.

Proceso

- Se crea una instancia de la base de datos SQLite3 utilizando la ruta especificada en la variable ***dbroute***.
- Se definen las variables nombreArchivo y extensión para el nombre y la extensión del archivo de texto que se generará.
- Se realiza una consulta a la base de datos para seleccionar todos los registros de la tabla ***mediciones***.
- En caso de que ocurra un error durante la consulta, se responde al cliente con un código de estado 500 y un mensaje de "Error interno del servidor".
- Se mapean las filas obtenidas de la base de datos para formatear los datos en un formato de texto que consiste en una línea por registro, separando cada valor por comas.
- Se escribe el contenido formateado en un archivo de texto con el nombre y extensión especificados, utilizando la función ***fs.writeFile***.
- Si ocurre un error durante la escritura del archivo, se responde al cliente con un código de estado 500 y un mensaje de "Error interno del servidor".
- Si la escritura del archivo es exitosa, se configura la respuesta al cliente con un código de estado 200 y se establece el encabezado ***Content-Disposition*** para indicar que se va a descargar un archivo.
- Se crea un flujo de lectura del archivo recién creado y se transmite como respuesta al cliente utilizando ***fs.createReadStream*** y ***pipe***.

- **Respuesta al cliente (Imagen 8.16) :**

Al usar el botón **Descargar Datos** que se encuentra en la página html (Imagen 8.15) se da como respuesta el archivo de texto formateado (Imagen 8.16).

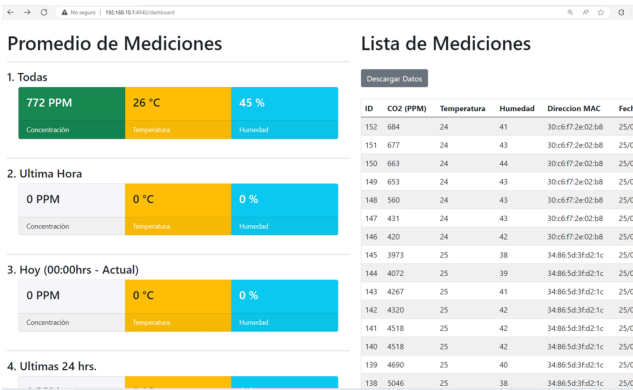


Imagen 8.15. Página html que incorpora el botón “Descargar Datos” para descargar un archivo de texto que contiene todos los datos recopilados por los sensores.

```
122,429,27,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:22:15
123,429,27,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:22:26
124,429,27,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:22:36
125,429,27,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:22:46
126,429,27,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:22:57
127,429,26,42,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:23:07
128,431,27,48,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:33:33
129,434,27,48,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:33:44
130,437,27,48,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:34:04
131,437,27,49,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:34:15
132,437,27,48,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 13:34:34
133,4072,25,37,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:26:34
134,3712,25,37,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:26:45
135,3537,25,37,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:26:58
136,4132,25,37,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:27:08
137,4858,25,37,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:27:19
138,5046,25,38,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:27:29
139,4690,25,40,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:27:40
140,4518,25,42,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:27:50
141,4518,25,42,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:28:01
142,4320,25,42,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:28:11
143,4267,25,41,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:28:21
144,4072,25,39,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:28:32
145,3973,25,38,192.168.10.2,34:86:5d:3f:d2:1c,25/01/2024, 14:28:42
146,420,24,42,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 14:29:19
147,431,24,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 14:29:29
148,560,24,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 14:29:40
149,653,24,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 14:29:50
150,663,24,44,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 14:30:01
151,677,24,43,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 14:30:11
152,684,24,41,192.168.10.2,30:c6:f7:2e:02:b8,25/01/2024, 14:30:22
```

Imagen 8.16. Archivo de texto formateado proporcionado por el servidor.

Errores y Excepciones

- Si ocurre algún error durante la ejecución de la función, se captura y se emite al cliente.

Funciones Complementarias

constantes.js

Las siguientes funciones son dadas ya que sirven como auxiliar para obtener la lista de dispositivos ESP32 conectados a la red 'interna' del Raspberry Pi. De tal manera que nos sirven para llevar a cabo el registro de las mediciones en la tabla de 'mediciones' de la base de datos (airhealthy).

1. Función: obtenerTablaArp(dev)

Esta función ejecuta el comando **arp** de linux pero con ayuda de la biblioteca de Node.js llamada **arp-a**, el cual retorna la lista de dispositivos en la tabla ARP del sistema para obtener las direcciones ipv4 correspondiente a cada uno de ellos. Luego, procesa esta salida para crear un array de objetos que contienen detalles como la dirección ipv4, dirección mac y la interfaz de red (del servidor) que en este caso pertenece a la red con nombre **interna** (Imagen 9.1).

La función posee la siguiente forma:

```
var tbl= [];  
  
function obtenerTablaArp(dev){  
  return new Promise(function(resolve, reject){  
  
    arp.table(function(error, entrada){  
      if(error){  
        reject(error);  
        return;  
      }  
  
      // Cuando se termine la lista..  
      if(entrada == null){  
        resolve();  
        return;  
      }  
  
      if(entrada.iface == dev){  
        tbl.push(entrada);  
      }  
    });  
  });  
}
```

Imagen 9.1. Proceso de la función obtenerTablaArp().

Argumentos

- **dev** → parámetro de la función que permite decirle a la función **table()** de la biblioteca **arp-a** que obtenga la lista de dispositivos pero de cierta red.. En este caso, la red con nombre **interna** del raspberry.

Salida

Devuelve un array de objetos que contienen detalles como la dirección ipv4, dirección mac y la interfaz de red (*interna*).

Errores

Si ocurre un error al ejecutar el comando, se devolverá el resultado del error a la consola del sistema..

2. Función: obtenerNombreHost(ip)

Esta función permite obtener el nombre del host, es decir, del dispositivo que corresponde a la ip proporcionada a la función. Por ejemplo, si se pasa un valor como “192.168.10.178” la función devolverá “esp32-XXXXX”, cuyo nombre es el hostname del dispositivo que posee esa dirección ipv4 (Imagen 9.2).

```
function obtenerNombreHost(ip){
  return new Promise(function(resolve, reject){
    dns.reverse(ip, (err, nombreHost) => {
      if(err){
        resolve(null);
      }else{
        resolve(nombreHost);
      }
    });
  });
}
```

Imagen 9.2. Proceso que realiza la función obtenerNombreHost() para recuperar el hostname dada una dirección ipv4.

Ejemplo de Uso (Imagen 9.3):

```
const hostname= await obtenerNombreHost(ip);
```

Imagen 9.3. Manera en que se hace uso de la función obtenerNombreHost().

Argumentos

- **ip** → la dirección ipv4 de un dispositivo.

Salida

Devuelve el nombre del host del dispositivo correspondiente a la dirección ipv4 proporcionada.

Errores

Si ocurre un error al intentar recuperar el nombre del host se devolverá el valor *null*.

3. Función: lista_arp(dev)

Esta función tiene la finalidad de obtener una lista de direcciones IP y sus correspondientes nombres de host (si están disponibles) en una red local específica (Imagen 9.4).

```
const lista_arp = async function(dev){
  try{
    await obtenerTablaArp(dev);

    for(var i=0; i < tbl.length; i++){
      var ip = tbl[i].ip;

      const hostname= await obtenerNombreHost(ip);

      if( hostname ){
        tbl[i].address = hostname[0];
      }else{
        tbl.splice(i, 1);
      }
    }

    return tbl.filter((objeto) => objeto.address !== undefined);

  }catch(error){
    console.error( error );
  }
};
```

Imagen 9.4. Algoritmo de la función lista_arp().

Ejemplo de Uso (Imagen 9.5):

```
var arp = await lista_arp('interna');
```

Imagen 9.5. Manera en que se hace uso de la función lista_arp().

Argumentos

- **dev** → parámetro de la función que permite decirle a la función **table()** de la biblioteca **arp-a** que obtenga la lista de dispositivos pero de cierta red.. En este caso, la red con nombre **interna** del raspberry.

Salida

Devuelve un arreglo de objetos (con formato json) que contiene detalles como: dirección ipv4, dirección mac, hostname y el nombre de la interfaz de la red de donde provienen estas listas de dispositivos.

Errores

Si ocurre un error se imprimirá como mensaje en la consola del sistema.

4. Función: lista_medidores()

Esta función hace uso finalmente de **lista_arp()** la cual una vez recuperada toda la información junto con sus nombres de host, filtra los datos con la condición de que el nombre del host corresponda al de un ESP32.

```
const lista_medidores = async function(devices= null){  
    var arp = await lista_arp('interna');  
    if( devices === null || devices == 'all' ){  
        arp = arp.filter(obj => obj.address.toLowerCase().indexOf('esp') !== -1 && obj.address.toLowerCase().indexOf('esp32') !== -1);  
    }  
    return arp;  
};
```

Imagen 9.6. Proceso que realiza la función lista_medidores() para obtener la lista de microcontroladores ESP32 conectados al servidor raspberry.

Ejemplo de Uso:

```
const esp32Devices = await funciones.lista_medidores();
```

Imagen 9.7. Manera en que se hace uso de la función lista_medidores().

Argumentos

Ninguno.

Salida

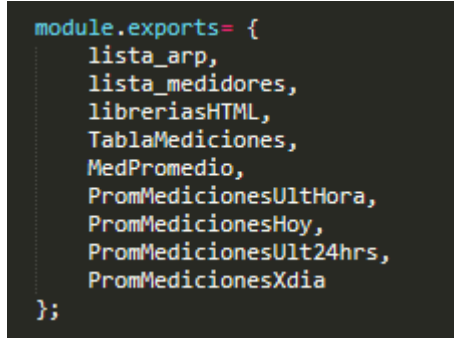
Devuelve un array de objetos que corresponde a la lista de dispositivos ESP32 conectados al servidor (en ese momento).

Errores

Si no existen dispositivos ESP32 conectados al servidor se devolverá el arreglo vacío.

5. Exportación de las Funciones

Finalmente, dadas las condiciones sintácticas de trabajar en NodeJS, para cada archivo independiente se debe realizar la exportación de las funciones para que de esta manera se pueda hacer uso de ellas en otros archivos del proyecto.

A screenshot of a code editor showing a JavaScript module export. The code is as follows:

```
module.exports = {  
  lista_arp,  
  lista_medidores,  
  libreriasHTML,  
  TablaMediciones,  
  MedPromedio,  
  PromMedicionesUltHora,  
  PromMedicionesHoy,  
  PromMedicionesUlt24hrs,  
  PromMedicionesXdia  
};
```

The code is written in a dark-themed editor with syntax highlighting: 'module' is blue, 'exports' is green, and the rest is white. The closing brace and semicolon are on the same line as the last property.

Imagen 9.8. Manera en que se exportan las funciones en un archivo en un entorno NodeJS.

Se exportan todas las funciones que se harán uso en el archivo **socket.js** del proyecto (Imagen 9.8).