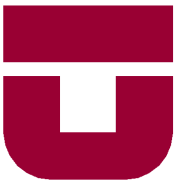


UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE COMPUTACIÓN

INFORME PROYECTO 1 TENDENCIAS EN TECNOLOGIAS COMPUTACIONALES.”

AUTORES:

Christian Lizama
Diego Aguilera

CURICÓ - CHILE
MES 3 DE 2023

Índice

Introducción 1
Marco teórico 2
Implementación 3
- **Esquema de BD en mongo** 3
- **Esquema de circuito antes de montarlo** 4
- **Circuito montado en la placa** 5
- **Explicación y descripción de los segmentos de código más importantes** 5
- **Parte server** 5
- **Parte cliente** 7

Presentación de resultados obtenidos 8
Conclusión 9
Referencias 10
Anexo 11
- **Código Arduino:** 11
- **Código de app.js:** 12
- **Código de alarmaController.js:** 13

Introducción

Este informe describe la implementación básica de un sistema que registra las mediciones de un sensor de movimiento en un ESP8266 y las envía a un servidor. El objetivo es presentar estas mediciones en forma de tabla simple en un servidor web y proporcionar una interfaz gráfica para visualizar gráficos y estadísticas de las mediciones. El sistema utiliza el framework MEVN (MongoDB, Express, Vue.js, Node.js) para construir el servidor.

El problema a resolver es la necesidad de monitorizar los movimientos en un área determinada en tiempo real y poder visualizar esta información en una interfaz gráfica. La solución consiste en utilizar un sensor de movimiento para detectar los movimientos, un ESP8266 para enviar las mediciones al servidor y un servidor web para almacenar y visualizar los datos.

Los objetivos específicos para resolver este problema son los siguientes:

Implementar el código necesario para registrar las mediciones del sensor en un ESP8266 y enviarlas al servidor.
Implementar un servidor que reciba los datos del ESP8266, para esto utilizamos una base de datos con MongoDB y los presentamos en forma de tabla simple en un servidor web utilizando el framework MEVN.
Desarrollar una interfaz gráfica que permita consultar estadísticas varias y visualizar gráficos de las mediciones almacenadas en el servidor.
En resumen, este informe describe la implementación de un sistema que utiliza un sensor de movimiento, un ESP8266 y un servidor web para monitorizar y visualizar los movimientos en tiempo real. El sistema ofrece una solución eficiente y escalable para la monitorización de movimiento en diferentes entornos.

Marco teórico

En este proyecto se utilizan diferentes tecnologías, microcontroladores, sensores y actuadores. A continuación, se detalla brevemente cada uno de ellos:

Microcontrolador ESP8266: es un microcontrolador de bajo costo y bajo consumo de energía que cuenta con conectividad Wi-Fi. Es muy utilizado en proyectos de IoT (Internet de las cosas) debido a su capacidad de conectarse a Internet y enviar y recibir datos. En este proyecto, se utiliza el ESP8266 para registrar las mediciones del sensor de movimiento y enviarlas al servidor.

Sensor de movimiento: es un sensor que detecta la presencia de movimiento en un área determinada. En este proyecto, se utiliza un sensor de movimiento para detectar movimientos en una determinada área y registrar la fecha y hora en que se produce el movimiento.

Actuador: Es un dispositivo que actúa en respuesta a una señal. En este proyecto, se utiliza un mini parlante para producir un sonido de pitido cada vez que el sensor detecta un movimiento.

Lenguaje de programación: en este proyecto, se utiliza el lenguaje de programación JavaScript para programar tanto el ESP8266 como el servidor web. También se utiliza HTML y CSS para desarrollar la interfaz gráfica del servidor web.

Framework MEVN: Es un conjunto de tecnologías que incluye MongoDB, Express, Vue.js y Node.js, utilizados para construir el servidor web en este proyecto. MongoDB es una base de datos NoSQL que se utiliza para almacenar las mediciones del sensor, Express es un framework de Node.js que se utiliza para construir la API RESTful, Vue.js es un framework de JavaScript que se utiliza para desarrollar la interfaz de usuario, y Node.js es un entorno de tiempo de ejecución de JavaScript que se utiliza para ejecutar el servidor.

En conclusión, el proyecto utiliza tecnologías y dispositivos comunes en proyectos de IoT, como el microcontrolador ESP8266, el sensor de movimiento y un mini parlante, y se utiliza el lenguaje de programación JavaScript y el framework MEVN para desarrollar la aplicación.

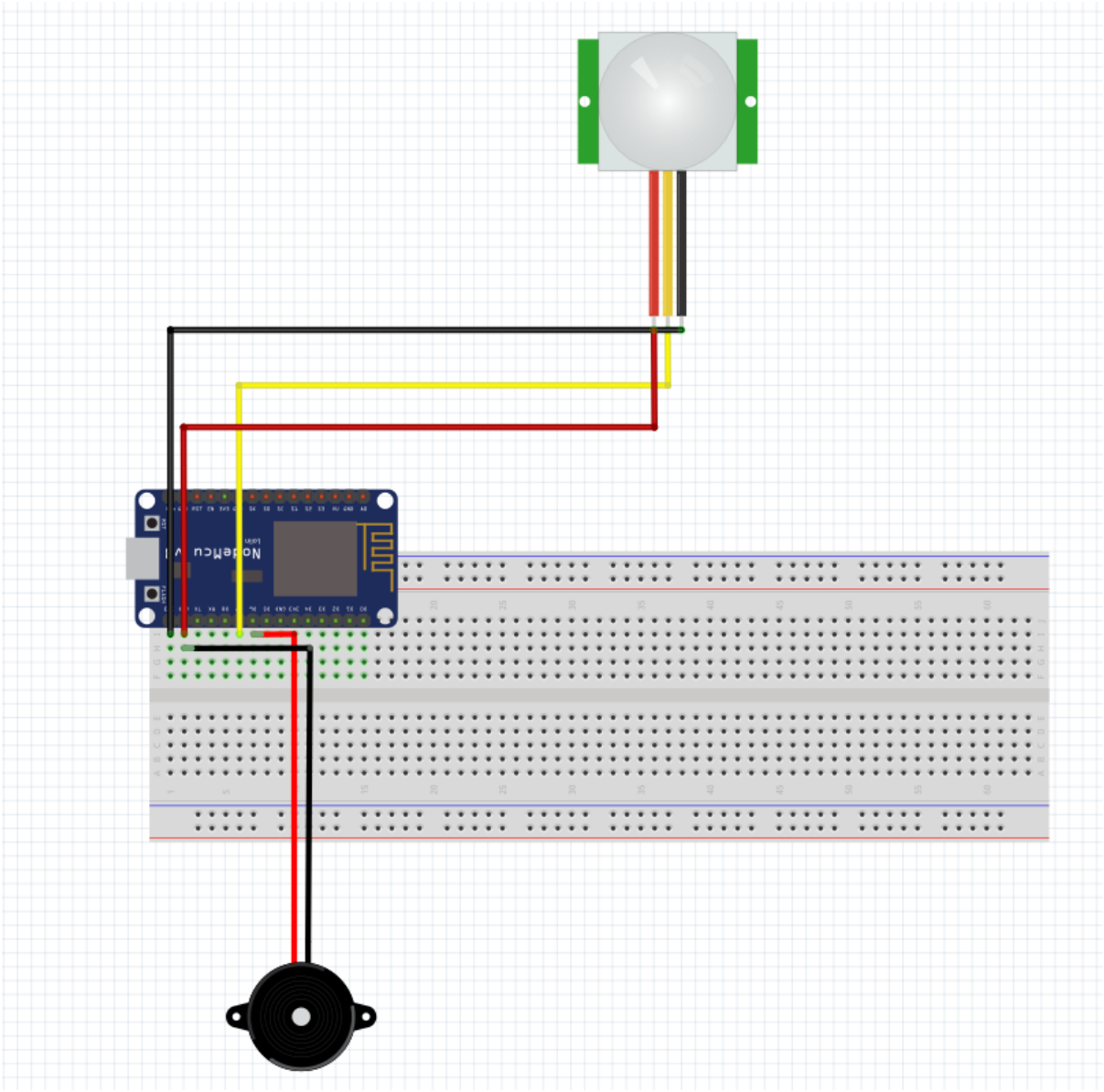
Implementación

Esquema de BD en mongo

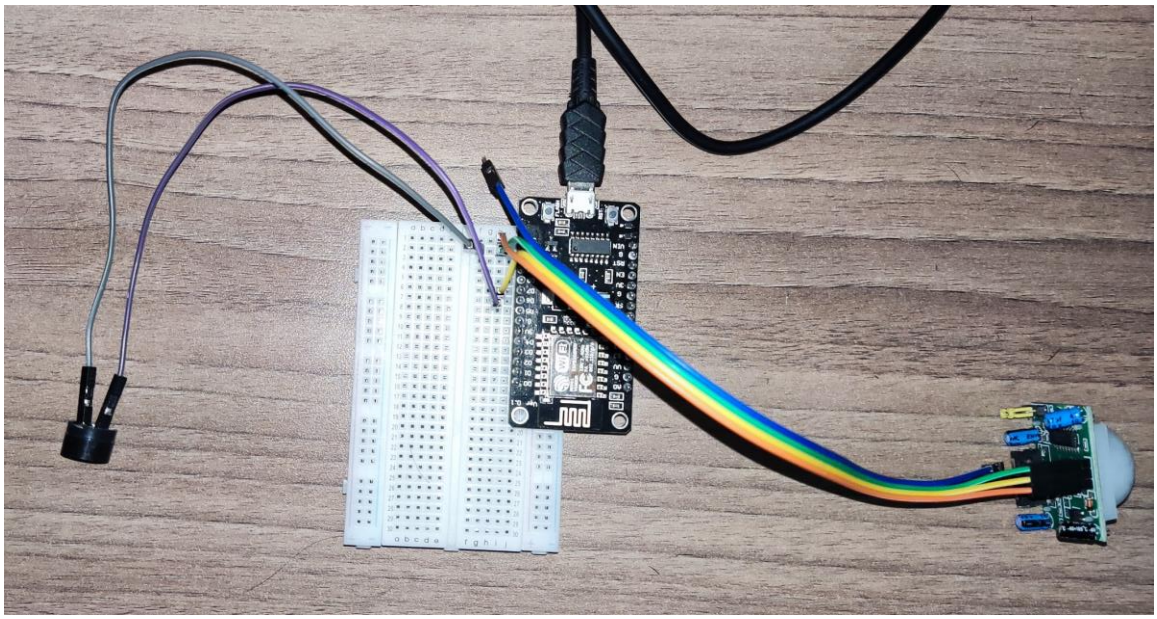
```
const alarmaSchema = new Schema({
    fecha: {type: String, required: [true, 'Fecha obligatoria']},
    tiempo: {type: String, required: [true, 'Tiempo obligatorio']},
    hora : {type: String, required: [true, 'Hora obligatoria']},
    minuto : {type: String, required: [true, 'Minuto obligatorio']},
    segundo : {type: String, required: [true, 'Segundo obligatorio']}
});
```

Donde podemos ver que una hora en formato de hora:minutos:segundos fue separada en 3 partes para poder generar gráficos de manera más simple, sin embargo, se dejó de igual manera la hora de forma completa, que en este caso corresponde al atributo tiempo, cabe resaltar que este se guardó como un string para poder mostrar el horario con formato de Chile de manera más simple y así evitar transformaciones por el lado del cliente.

Esquema de circuito antes de montarlo



Circuito montado en la placa



Explicación y descripción de los segmentos de código más importantes

Parte server

app.js: Este código es una aplicación web construida en Node.js y Express.js que se encarga de recibir datos de un sensor PIR (sensor de movimiento) a través de un puerto serial, y guardar la información recibida en una base de datos MongoDB.

En la primera parte del código se importan las diferentes dependencias necesarias para el funcionamiento de la aplicación, como Express.js, Morgan, Cors, Mongoose, entre otros. Además, se definen las rutas de los diferentes endpoints de la aplicación.

Luego, se establece la conexión con la base de datos MongoDB, utilizando la variable de entorno MONGO_URL definida en el archivo.env.

A continuación, se establece la conexión con el puerto serial del Arduino, y se crea un parser que convierte los datos recibidos en un formato legible para la aplicación.

Luego, se define una función que escucha los datos recibidos del sensor PIR, convierte el valor a un número entero, y si es igual a 1, guarda la fecha y hora actual en la base de datos utilizando la librería Moment.js.

Finalmente, se crea el servidor web utilizando Express.js, se establece el puerto en el que el servidor escuchará las peticiones, y se inicia el servidor.

alarmaController.js: Este código contiene tres métodos para hacer consultas a una base de datos MongoDB utilizando el modelo de datos "alarma". El primer método llamado "query" busca todos los registros en la base de datos utilizando el método "find" del modelo "alarma". Si se encuentran registros, los devuelve en una respuesta HTTP 200, de lo contrario, devuelve una respuesta HTTP 404 indicando que no se encontraron registros.

El segundo método llamado "queryDia" busca el número de registros que existen para una fecha específica que es recibida en el cuerpo de la petición HTTP. Utiliza el método "find" del modelo "alarma" con una condición que filtra por fecha. Si se encuentran registros, los devuelve en una respuesta HTTP 200, de lo contrario, devuelve una respuesta HTTP 404 indicando que no se encontraron registros.

El tercer método llamado "queryHora" busca el número de registros que existen para cada hora de un día específico que es recibido como un parámetro de consulta en la petición HTTP. Utiliza un bucle para iterar sobre cada hora del día y utiliza el método "find" del modelo "alarma" con una condición que filtra por fecha y hora. El número de registros encontrados para cada hora se agrega a un arreglo que se devuelve como una respuesta HTTP 200. Si no se encuentran registros para un día específico, devuelve un arreglo vacío. Si ocurre un error, devuelve una respuesta HTTP 500 indicando que ocurrió un error.

routes/alarma.js: Este código define las rutas para manipular los datos de alertas. El archivo comienza importando los módulos necesarios y definiendo un router.

Luego se define una ruta con el método HTTP GET para la raíz de la ruta "/".

A continuación, se definen otras tres rutas para los métodos GET "/getAlertas", "/queryDia" y "/queryHora". Cada una de estas rutas llama a diferentes funciones exportadas por el archivo "alarmaController.js" cuando se realiza una petición HTTP a ellas.

Finalmente, el archivo exporta el router para que pueda ser utilizado en otros archivos de la aplicación.

Parte cliente

main.js: Este código es el punto de entrada de una aplicación Vue.js. Aquí se importan y configuran varias dependencias de la aplicación, como Vue, Vuex, Vuetify y Axios.

La configuración de Axios es importante, ya que establece la URL base del servidor para realizar peticiones HTTP. Esta URL se obtiene de la variable de entorno VUE_APP_SERVER_URL definida en el archivo.env del proyecto. La URL se establece como valor por defecto de Axios, por lo que todas las peticiones realizadas con Axios utilizarán esta URL como base.

Finalmente, se crea una nueva instancia de Vue, que utiliza el componente principal de la aplicación (App.vue), así como el enrutador (router) y la instancia de Vuex (store). Esta instancia de Vue se monta en un elemento HTML con el ID "app".

GraficoDía.vue: Este es un componente de Vue.js que muestra un gráfico de barras de la cantidad de movimientos registrados en una determinada fecha. El gráfico se genera utilizando la biblioteca Chart.js y se actualiza según la fecha seleccionada por el usuario.

La plantilla del componente muestra un campo de texto con un ícono de calendario que abre un selector de fecha cuando se hace clic. Debajo del campo de fecha, se muestra el gráfico de barras.

El componente utiliza la biblioteca axios para hacer una solicitud GET a un endpoint '/alerta/queryHora' para obtener los datos necesarios para generar el gráfico. Los datos se almacenan en el objeto de datos 'chartData' y se actualizan cada vez que el usuario selecciona una fecha diferente.

El componente también incluye una función para formatear la fecha en un formato legible para el usuario y una serie de opciones de configuración para el selector de fecha y el gráfico de barras.

TablaMovimientos.vue: Este código es un componente Vue que contiene una tabla de datos que se puede filtrar, ordenar y paginar. La tabla muestra información de alertas y se comunica con un servidor a través de la biblioteca axios para obtener los datos.

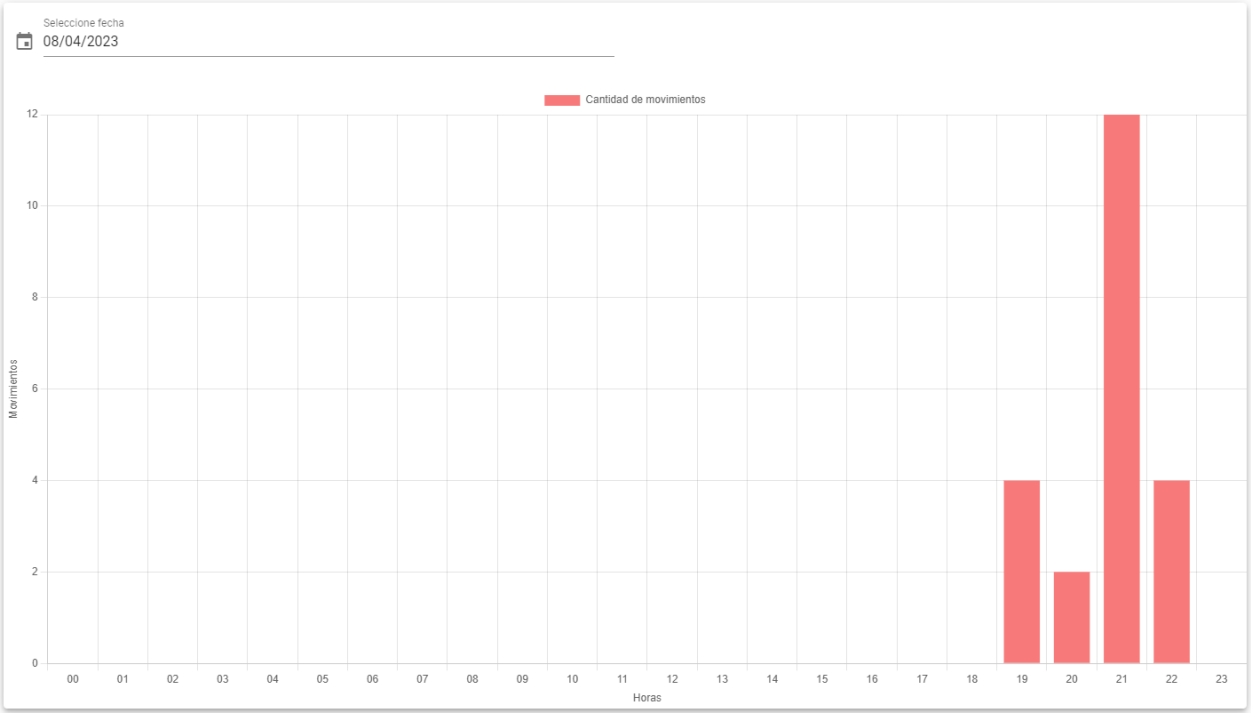
El componente utiliza una plantilla que contiene un contenedor que envuelve el componente v-data-iterator de Vuetify. El componente v-data-iterator es el que maneja la paginación, filtrado y ordenamiento de los datos. También contiene un formulario de búsqueda y un menú desplegable para ordenar los datos. La plantilla también contiene una ranura predeterminada que define cómo se visualiza cada elemento de la lista.

El componente tiene un objeto de datos que contiene las variables utilizadas para configurar el comportamiento del componente, como la cantidad de elementos por página, el criterio de ordenamiento y los datos a mostrar. También hay una función creada que se ejecuta cuando se carga el componente y utiliza axios para obtener los datos del servidor.

El componente también tiene varias funciones de ayuda que manejan el cambio de página, el cambio de cantidad de elementos por página y la actualización de los datos obtenidos del servidor.

Presentación de resultados obtenidos

El análisis de los datos recolectados muestra que, durante un día específico, se registraron movimientos en el área monitoreada. En particular, se observó que desde las 19:00 hasta las 22:00 horas se produjeron movimientos, como se muestra en el gráfico de barras correspondiente. Este resultado proporciona información valiosa sobre los patrones de actividad en el área monitoreada y puede ser utilizado para mejorar la eficiencia en la toma de decisiones en futuras situaciones similares.



Conclusión

En este proyecto se ha implementado con éxito un sistema para la monitorización de movimiento en tiempo real utilizando un sensor de movimiento, un ESP8266 y un servidor web. El sistema ofrece una solución eficiente y escalable para la monitorización de movimiento en diferentes entornos.

El sistema ha sido probado y se ha demostrado que es capaz de registrar las mediciones del sensor de movimiento y enviarlas al servidor de manera confiable. Además, se ha desarrollado una interfaz gráfica para visualizar gráficos y estadísticas de las mediciones almacenadas en el servidor.

Sin embargo, hay algunas áreas en las que se pueden realizar mejoras en el sistema. Por ejemplo, se podría considerar el uso de un sensor más preciso o la adición de más sensores para monitorear una mayor cantidad de áreas. También se podría mejorar la interfaz de usuario, agregando más funcionalidades como notificaciones en tiempo real y la posibilidad de exportar los datos a otros formatos.

En cuanto al rendimiento del sistema, se ha demostrado que es capaz de manejar grandes cantidades de datos sin problemas. Sin embargo, para mejorar la eficiencia del sistema, se podría implementar una técnica de compresión de datos antes de enviarlos al servidor.

En conclusión, el sistema desarrollado es una solución eficaz y escalable para la monitorización de movimiento en tiempo real. Si bien hay algunas áreas en las que se pueden realizar mejoras, en general el sistema cumple con los objetivos establecidos y puede ser utilizado en una amplia variedad de aplicaciones de monitoreo de movimiento.

Referencias

https://serialport.io/docs/
https://vue-chartjs.org/guide/
https://v2.vuetifyjs.com/en/
https://support.arduino.cc/hc/en-us/articles/4403050020114-Troubleshooting-PIR-Sensor-and-sensitivity-adjustment
https://momentjs.com/

Anexo

Código Arduino:

```
//Pines conectados a la placa
#define PIR_PIN D7
#define BUZZ_PIN D6

void setup() {
  Serial.begin(9600);
  pinMode(PIR_PIN, INPUT);
  pinMode(BUZZ_PIN, OUTPUT);
}

void loop() {
  //Leemos el valor del pin del sensor
  int Valor = digitalRead(PIR_PIN);
  //Si se detecta movimiento
  if(Valor==HIGH){
    //Hacemos sonar el buzzer
    tone(BUZZ_PIN, 440);
    //Enviamos el valor
    Serial.println(Valor);
  }
  else{
    //Si no se detecta movimiento dejamos de emitir ruido
    noTone(BUZZ_PIN);
  }
  //Usamos un delay de 1500 para poder evitar detectar falsos movimientos
  delay(1500);
}
```

Código de app.js:

```
//Conexion DB
const uri = process.env.MONGO_URL;
const options = { useNewUrlParser: true };

mongoose.connect(uri, options).then(
  (client) => {
    console.log("Conectado a DB");
  },
  (err) => {
    console.log(err);
  }
);

//Puerto serial del usb
const puertoSerial = "COM4";
// Crear la conexión serial con Arduino
const port = new SerialPort({ path: puertoSerial, baudRate: 9600 });
const parser = port.pipe(new ReadlineParser({ delimiter: "\n" }));

// Escuchar los datos del sensor PIR
parser.on("data", (data) => {
  const pirValue = parseInt(data); // Convertir el valor recibido a número entero
  console.log("Valor del sensor PIR:", pirValue);
  if(pirValue==1){
    const fechaChile = moment().utcOffset('-04:00');
    const fecha = fechaChile.format("DD/MM/YYYY");
    const tiempo = fechaChile.format("HH:mm:ss");
    const hora = fechaChile.format("HH");
    const minuto = fechaChile.format("mm");
    const segundo = fechaChile.format("ss");
    const nuevaAlerta = alarma({fecha: fecha,tiempo: tiempo,hora:hora,minuto:minuto,segundo:segundo})
    nuevaAlerta.save().then(()=>{
        console.log("Nueva alerta guardada en la bd")
    }).catch((err) => {
        console.error('Error al guardar la fecha en MongoDB:', err);
    })
  }
});
```

Código de alarmaController.js:

```
//Metodo para obtener todas las alarmas
const query = async (req, res, next) => {
  try {
    const reg = await alarma.find();
    if (!reg) {
      res.status(404).send({
        message: "No existe ningún registro",
      });
    } else {
      res.status(200).json(reg);
    }
  } catch (e) {
    res.status(500).send({
      message: "Ocurrio un error",
    });
    next(e);
  }
};

//Metodo para contar todas las alarmas de un dia
const queryDia = async (req, res, next) => {
  try {
    const { dia } = req.body;
    const reg = await alarma.find({ fecha: dia }).count();
    if (!reg) {
      res.status(404).send({
        message: "No existe ningún registro",
      });
    } else {
      res.status(200).json(reg);
    }
  } catch (e) {
    res.status(500).send({
      message: "Ocurrio un error",
    });
    next(e);
  }
};
```

```
//Metodo para contar las alarmas de un dia en las 24 hrs
const queryHora = async (req, res, next) => {
  try {
    const { dia } = req.query;
    console.log(dia)
    var arreglo = [];

    // Llenar el arreglo con números en formato de 00 a 23
    for (var i = 0; i <= 23; i++) {
      // Convertir el número a una cadena de texto y agregarle un 0 al principio si es menor a 10
      var numero = i < 10 ? "0" + i : "" + i;
      const conteo = await alarma.find({ fecha: dia, hora: numero }).count();
      // Agregar el número al arreglo
      arreglo.push(conteo);
    }

    res.status(200).send(arreglo);
  } catch (e) {
    res.status(500).send({
      message: "Ocurrio un error",
    });
    next(e);
  }
};
```

Para más información sobre el código visitar el enlace del repositorio GitHub:

- https://github.com/christianLizama/ProyectoloT-Arduino