Code of the server:

require('dotenv').config(); // Load environment variables from .env file

const express = require('express'); // Import Express framework

const cors = require('cors'); // Import CORS middleware

const http = require('http'); // Import Node's HTTP module

const { Server } = require("socket.io"); // Import Socket.IO Server class

const { SerialPort } = require('serialport')

const { ReadlineParser } = require('@serialport/parser-readline')

// Create an Express application

const app = express();

// Use CORS middleware to enable cross-origin requests

app.use(cors());

app.use(express.json()); // Middleware to parse JSON bodies

// Create an HTTP server using the Express app

const server = http.createServer(app);

/////sequelize ////////////////

//import {Sequelize, Model, DataTypes} from 'sequelize';

//import { Op } from 'sequelize';

const {Sequelize, Model, DataTypes, JSON} = require('sequelize');

const { Op } = require('sequelize');

const bodyParser = require('body-parser');

// Create Sequelize instance

const sequelize = new Sequelize({

  dialect: 'sqlite',

  storage: './database.sqlite'

});

class Params extends Model {}

Params.init({

  smoke: DataTypes.INTEGER,

  hum: DataTypes.INTEGER,

  Temp: DataTypes.INTEGER,

}, { sequelize, modelName: 'params' });

// Sync models with database

sequelize.sync();

// Middleware for parsing request body

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

app.get('/params', async (req, res) => {

  const params = await Params.findAll();

  //const myUser = JSON.stringify(users);

  res.send(params);

});

/////////////////////////////

// Initialize a new instance of Socket.IO by passing the HTTP server

const io = new Server(server, {

  cors: {

    origin: "http://localhost:3000", // Allow requests from this origin and my frontend port = 3001

    methods: ["GET", "POST"], // Allow these HTTP methods

  },

});

// Listen for incoming Socket.IO connections

io.on("connection", (socket) => {

  console.log("User connected ", socket.id); // Log the socket ID of the connected user

  // Listen for "send\_message" events from the connected client

  /\*socket.on("send\_message", (data) => {

      console.log("Message Received ", data); // Log the received message data

      // Emit the received message data to all connected clients

      //io.emit("receive\_message", data);

  });\*/

});

const port = new SerialPort({ path: 'COM5',baudRate: 9600,}, function (err) {

  if (err) {

    return console.log('Error: ', err.message)

  }

})

port.write('main screen turn on', function(err) {

  if (err) {

    return console.log('Error on write: ', err.message)

  }

  console.log('message written')

})

// Read data that is available but keep the stream in "paused mode"

port.on('readable', function () {

    //console.log('Data:', port.read())

    port.read();

  })

  // Switches the port into "flowing mode"

//let tem = 1;

function getRandomArbitrary(min, max) {

  return Math.random() \* (max - min) + min;

}

let s = '';

let h = '';

let t = '';

 /\* port.on('data', function (data) {

    // str = data.toString();

    // JSON.parse(str);

    //str = str.json();

     //str = JSON.stringify(str);

      //str1 = str.replace(/\r?\n|\r/g, "");

      //str2 = Object.create(str1);

      //str2 = JSON.parse(str);

      /\*const dist = str.slice(0,2);

      const vib = str.slice(2,4);

      const hum = str.slice(4,6);

      console.log(dist);

      console.log(vib);

      console.log(hum);

      io.emit("receive\_message", dist, vib,hum);

      Params.create({dist: dist, pwm:vib, hum:hum});\*/

       //const str1 = data.toString();

        //console.log(str1[0]);

      /\* const str1 = data.toString();

       const str = Number(str1);

      const str2 = str + getRandomArbitrary(28, 32);

       console.log(str2)

       Temprature.create({temp: str2});\*/

 // })

  const parser = port.pipe(new ReadlineParser({ delimiter: '\r\n' }))

    parser.on('data',(data) => {

      //console.log(data);

    switch (true) {

        case data.startsWith("Smoke"):

          //d = data.substr(data.lenght - 5);

          s = data.slice(6);

          break;

        case data.startsWith("Humi"):

         // h = data.substr(data.lenght - 2 );

          h = data.slice(5);

          break;

          case data.startsWith("Temp"):

         // h = data.substr(data.lenght - 2 );

          t = data.slice(5);

          break;

      }

      try{

        io.emit("receive\_message", s , h, t);

        Params.create({smoke:s, hum:h, Temp:t});

      }catch(e){

        console.log(e);

      }

      console.log(s);

      console.log(h);

      console.log(t);

      //io.emit("receive\_message", data);

    })

/\*var parser = new ReadlineParser()

port.pipe(parser)

parser.on('data', function (data) {

  //str = data.toString(); //Convert to string

 // str1 = str.replace(/\r?\n|\r/g, ""); //remove '\r' from this String

 // str2 = JSON.stringify(str); // Convert to JSON

  //str3 = JSON.parse(str2); //Then parse it;

 // let dist = str.substr(0,1);

  //io.emit("receive\_message", str);

  let arr = [];

  arr.push(data);

    console.log(data);

    console.log(arr);

})\*/

  const PORT = process.env.PORT || 5050; // Define a default port if PORT is not set in .env

  server.listen(PORT, () => {

      console.log("Server is running on port " + PORT);

    });

    // Pipe the data into another stream (like a parser or standard out)

  //const lineStream = port.pipe(new Readline())

Code of the dashboard:

import logo from './logo.svg';

import './App.css';

import React, { useEffect, useState, PureComponent } from "react";

import io from 'socket.io-client'; // Import the socket.io client library

import Stack from '@mui/material/Stack';

import Box from '@mui/material/Box';

import { SparkLineChart } from '@mui/x-charts/SparkLineChart';

import { LineChart } from '@mui/x-charts/LineChart';

import { BarChart } from '@mui/x-charts/BarChart';

import { Gauge } from '@mui/x-charts/Gauge';

import { GaugeComponent } from 'react-gauge-component';

import 'bootstrap/dist/css/bootstrap.css';

import { Container, Row, Col, ResponsiveEmbed,Toast,Button,ToastContainer } from 'react-bootstrap';

import { Sparklines, SparklinesLine, SparklinesSpots, SparklinesReferenceLine } from 'react-sparklines';

import Alert from 'react-bootstrap/Alert';

import Speech from 'react-speech';

// Establish a socket connection to the server at the specified URL

const socket = io.connect('http://localhost:5050');

function App()  {

  const [smk , setsmk] = useState();

  const [hum, sethum] = useState();

  const [arr ,setarr] = useState([]);

  const [temp ,settemp] = useState();

  const [smkarr, setsmkarr] = useState([]);

  const [humarr, sethumarr] = useState([]);

 const [v1 , setv1] = useState();

 const [v2 , setv2] = useState();

 const [s ,sets] = useState();

 const vm = 22.4;

 const m = 44;

 const [ pred , setpred] = useState();

 // let avr = [];

 const url = "http://localhost:5050/params";

 async function gettemp () {

  try {

    const response = await fetch(url);

    if (!response.ok){

      throw new Error(`res status : ${response.status}`);

    }

    const res = await response.json();

    //console.log( res[100].temp);

   for (let i = 0; i < res.length; ++i){

    settemp([...temp,res[i].temp]);

   }

      //const temp = res.map((item)=> item.temp.reduce((a,b)=> a + b) / res.length, res)

      //setcrdate(res.createdAt);

  } catch (error) {

    console.error(error);

  }

 }

  useEffect(() => {

    gettemp();

    // Listen for incoming messages from the server

    socket.on("receive\_message", (s,h,t) => {

      setsmk(s);

      sethum(h);

      settemp(Number(t));

      function getRandomInt(max) {

        return Math.floor(Math.random() \* max);

      }

     // setavrg(avg);

       if (arr.length >=  8){

          let avr = [...arr];

          let srr= [...smkarr];

          let hrr = [...humarr];

          avr.shift();

          srr.shift();

          hrr.shift();

          setarr(avr);

          setsmkarr(srr);

          sethumarr(hrr);

        }else{

            setarr([...arr, t]);

            arr.push(t);

            setsmkarr([...smkarr, s]);

            smkarr.push(s);

            sethumarr([...humarr,h]);

            humarr.push(h);

        }

        /\*const sum = humarr.sort((a,b) => a - b );

        const avg = (sum / humarr.length);

        setconfac(avg);\*/

    });

    // Cleanup the effect by removing the event listener when the component unmounts

    return () => {

      socket.off("receive\_message");

    };

  }, [arr]); // Empty dependency array ensures this runs only once when the component mounts

  const msgs = () => {

   const msg = new SpeechSynthesisUtterance()

   msg.text = "Be alert, the threshold has been exceeded"

   window.speechSynthesis.speak(msg)

  }

    const msgh = () => {

      const msg1 = new SpeechSynthesisUtterance()

      msg1.text = "Be alert, the threshold has been exceeded"

      window.speechSynthesis.speak(msg1)

    }

    const msgc = () => {

      const msg2 = new SpeechSynthesisUtterance()

      msg2.text = "Be alert, the threshold has been exceeded"

      window.speechSynthesis.speak(msg2)

    }

  const ih = Number(hum);

  const is = Number(smk);

  const it = Number(temp);

    const calc = () => {

    const  tv = ( Number(v1) - Number(v2)) / Number(vm);

     const ms = Number(m)/Number(s);

     const  tt = 273 / temp;

      const coa = smk \* .27

     let pr =  coa \* tv \* ms \* tt

      setpred(pr);

    }

  return (

    <>

    <Row>

      <Col md={6} className="mb-2">

      <ToastContainer

          className="p-3"

          position={'middle-start'}

          style={{ zIndex: 1 }} >

        <Toast >

          <Toast.Header>

            <strong className="me-auto">Smoke :</strong>

            <small>{is} ppm </small>

          </Toast.Header>

          <Toast.Body>

          {is > 99 ? <> <Alert variant='danger'> Be alert, the threshold has been exceeded</Alert> </> : ""}

          <Sparklines data={smkarr} margin={6}>

           <SparklinesLine style={{ strokeWidth: 3, stroke: "red", fill: "none" }} />

           <SparklinesSpots size={4} style={{ stroke: "#336aff", strokeWidth: 3, fill: "white" }} />

           <SparklinesReferenceLine type="min" />

          </Sparklines>

      </Toast.Body>

        </Toast>

        </ToastContainer>

      </Col>

    <Col md={6} className="mb-2">

    <ToastContainer

          className="p-3"

          position={'middle-center'}

          style={{ zIndex: 1 }}>

        <Toast >

          <Toast.Header>

            <strong className="me-auto">Temprature:</strong>

            <small>{temp} C</small>

          </Toast.Header>

          <Toast.Body>

          {it > 36? <> <Alert variant='danger'> Be alert, the threshold has been exceeded</Alert> </> : ""}

          <Sparklines data={arr} margin={6}>

           <SparklinesLine style={{ strokeWidth: 3, stroke: "red", fill: "none" }} />

           <SparklinesSpots size={4} style={{ stroke: "#336aff", strokeWidth: 3, fill: "white" }} />

           <SparklinesReferenceLine type="min" />

          </Sparklines>

      </Toast.Body>

        </Toast>

        </ToastContainer>

      </Col>

      <Col md={6} className="mb-2">

      <ToastContainer

          className="p-3"

          position={'middle-end'}

          style={{ zIndex: 1 }} >

        <Toast >

          <Toast.Header>

          <strong className="me-auto">humidity :</strong>

            <small>{hum} %</small>

          </Toast.Header>

          <Toast.Body>

          {ih < 60 ? <><Alert variant='danger'>Be alert, the threshold has been exceeded</Alert></>: ""}

          <Sparklines data={humarr} margin={6}>

           <SparklinesLine style={{ strokeWidth: 3, stroke: "red", fill: "none" }} />

           <SparklinesSpots size={4} style={{ stroke: "#336aff", strokeWidth: 3, fill: "white" }} />

           <SparklinesReferenceLine type="min" />

          </Sparklines>

      </Toast.Body>

      </Toast>

      </ToastContainer>

      </Col>

    </Row>

    <div style={{width: 150, marginLeft: 20}}>

      <Row>

        <Col>

        <input placeholder='v1' onChange={(e) => setv1(e.target.value)} />

        </Col>

      <Col>

      <input placeholder='v2' onChange={(e) => setv2(e.target.value)}  />

      </Col>

      </Row>

      <Row>

      <input placeholder='s'  onChange={(e) => sets(e.target.value)}  />

      </Row>

    <Button  onClick={calc}>calculate</Button>

    <p>prediction : {pred}</p>

    </div>

</>

  );

}

export default App;

/\*

 <Col md={6} className="mb-2">

      <ToastContainer

          className="p-3"

          position={'middle-start'}

          style={{ zIndex: 1 }} >

        <Toast >

          <Toast.Header>

            <strong className="me-auto">Temprature :</strong>

            <small>{receiveMessage} C</small>

          </Toast.Header>

          <Toast.Body>

          <Sparklines data={arr} margin={6}>

           <SparklinesLine style={{ strokeWidth: 3, stroke: "#336aff", fill: "none" }} />

           <SparklinesSpots size={4} style={{ stroke: "#336aff", strokeWidth: 3, fill: "white" }} />

           <SparklinesReferenceLine type="min" />

          </Sparklines>

      </Toast.Body>

        </Toast>

        </ToastContainer>

      </Col>

\*/

// <div style={{width: 400,marginLeft:250, marginTop: 300}}>

// <div  style={{display: 'flex', flexDirection: 'row' , width: '100%', textAlign: 'center', fontSize:50}}>

/\*        <SparkLineChart

          data={arr}

          xAxis={{

            scaleType: 'time',

            data: [

              new Date(2022, 5, 1),

              new Date(2022, 5, 2),

              new Date(2022, 5, 5),

              new Date(2022, 5, 6),

              new Date(2022, 5, 7),

              new Date(2022, 5, 8),

              new Date(2022, 5, 11),

              new Date(2022, 5, 12),

            ],

            valueFormatter: (value) => value.toISOString().slice(0, 10),

          }}

          height={100}

          showTooltip

          showHighlight

        />\*/

        /\*

        const kbitsToMbits = (value) => {

      return value + ' m/s';

  }

        <GaugeComponent

            type="radial"

              arc={{

                subArcs: [

                  {

                    limit: .020,

                    color: '#5BE12C',

                    showTick: true,

                  },

                  {

                    limit: .040,

                    color: '#F5CD19',

                    showTick: true

                  },

                  {

                    limit: .060,

                    color: '#F58B19',

                    showTick: true

                  },

                  {

                    limit: .1,

                    color: '#EA4228',

                    showTick: true

                  },

                ]

              }}

              pointer={{

                elastic: true,

                animationDelay: 0

              }}

              labels={{

                valueLabel: {

                  style: {fontSize: 40},

                  formatTextValue: kbitsToMbits

                },}}

              value={vib}

              minValue={0}

              maxValue={.1}

            />\*/

            /\*

             <GaugeComponent

              type="semicircle"

              arc={{

                colorArray: ['#00FF15', '#FF2121'],

                padding: 0.02,

                subArcs:

                  [

                    { limit: 40 },

                    { limit: 60 },

                    { limit: 70 },

                    {},

                    {},

                    {},

                    {}

                  ]

              }}

              pointer={{type: "blob", animationDelay: 0 }}

              value={hum}

            />\*/

Code of Arduino UNO:

#include <DHT.h>;

#define DHTPIN 2     // what pin we're connected to

#define DHTTYPE DHT22   // DHT 22  (AM2302)

DHT dht(DHTPIN, DHTTYPE); //// Initialize DHT sensor for normal 16mhz Arduino

#include <Wire.h>

/\* Change the threshold value with your own reading \*/

#define Threshold 400

#define MQ2pin 0

float sensorValue;  //variable to store sensor value

int hum;  //Stores humidity value

int temp; //Stores temperature value

const int buzzer = 11; //buzzer to arduino pin

const int sensor = 7; //signal pin of sensor to digital pin .

 int state = LOW;

 int val = 0;

void setup() {

  pinMode(sensor, INPUT); // PIR motion sensor is determined is an input here.

  Serial.begin(9600); // sets the serial port to 9600

  Wire.begin();

  pinMode(buzzer, OUTPUT); // Set buzzer - pin 9 as an output

  Serial.println("MQ2 warming up!");

  delay(2000); // allow the MQ2 to warm up

  dht.begin();

}

void loop() {

  sensorValue = analogRead(MQ2pin); // read analog input pin 0

  Serial.print("Smoke: ");

  Serial.println(sensorValue);

  /\*if(sensorValue > Threshold)

  {

    Serial.print(" | Smoke detected!");

  }

  Serial.println("");\*/

  //Read data and store it to variables hum and temp

    hum = dht.readHumidity();

    temp= dht.readTemperature();

    if(hum<60)

    {

      tone(buzzer, 1000); // Send 1KHz sound signal...

      delay(1000);         // ...for 1 sec

       noTone(buzzer);

      delay(1000);

    }

    if(temp > 36)

    {

      tone(buzzer, 3000); // Send 1KHz sound signal...

      delay(1000);         // ...for 1 sec

      noTone(buzzer);

      delay(1000);

    }

    Serial.print("Humi:");

    Serial.println(hum);

    Serial.print("Temp:");

    Serial.println(temp);

   /\* tone(buzzer, 3000); // Send 1KHz sound signal...

    delay(1000);         // ...for 1 sec

    noTone(buzzer);     // Stop sound...

    delay(1000);  \*/       // ...for 1sec

    delay(2000); // wait 2s for next reading

}