**The 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

// to make the server on express template using node js , to first made it and give the server link

const { Server } = require("socket.io"); // Import Socket.IO Server class // take the data from arduino and send it to the dashbord ,

//the another socket in the dash bord recieve the data from the server

const { SerialPort } = require('serialport')// take the readings from the arduino to the server

const { ReadlineParser } = require('@serialport/parser-readline') // display the readings taken from the arduino

// 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);

// data base

/////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({

  dist: DataTypes.INTEGER,

  pwm: DataTypes.INTEGER,

  hum: DataTypes.INTEGER,

  Temp: DataTypes.INTEGER,

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

///// end of the data base

// 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);

  });\*/

});

//connect with arduino

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 d = '';

let v = '';

let h = '';

let t = '';

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

    // str = data.toString();

       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("Distance"):

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

          d = data.slice(9);

          break;

        case data.startsWith("vib"):

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

          v= data.slice(4);

          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", d, v , h, t);

        Params.create({dist: d, pwm:v, hum:h, Temp:t});

      }catch(e){

        console.log(e);

      }

      console.log(d);

      console.log(v);

      console.log(h);

      console.log(t);

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

    })

/\*var parser = new ReadlineParser()

port.pipe(parser)

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

})\*/

  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())

**The code of the dash board:**

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 [receiveMessage, setReceiveMessage] = useState("");

  const [vib , setVib] = useState("");

  const [hum, sethum] = useState("");

  const [arr ,setarr] = useState([]);

  const [temp ,settemp] = useState([]);

  const [avrg, setavrg] = useState();

  const [vibarr, setvibarr] = useState([]);

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

  const[crdate, setcrdate] = useState([]);

  const [ crks, setcrks] = useState('');

  const [lifespan , setlifespan] = useState("");

  const [confac , setconfac] = useState("");

  const [ extfac, setextfac] = useState("");

  const [ predicted, setpredicted] = useState();

  const [bage , setbage] = 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", (d,v,h,t) => {

      setReceiveMessage(t);

      setVib(v);

      sethum(h);

      setcrks(d);

      function getRandomInt(max) {

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

      }

      //setavrg(avg);

       if (arr.length >=  8){

          let avr = [...arr];

          let vrr= [...vibarr];

          let hrr = [...humarr];

          avr.shift();

          vrr.shift();

          hrr.shift();

          setarr(avr);

          setvibarr(vrr);

          sethumarr(hrr);

        }else{

            setarr([...arr, t]);

            arr.push(t);

            setvibarr([...vibarr, v]);

            vibarr.push(v);

            sethumarr([...humarr,h]);

            humarr.push(h);

        }

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

        const avg = (sum / humarr.length)/10;

        setextfac(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 msgv = () => {

   const msg = new SpeechSynthesisUtterance()

   msg.text = "Warning warning vibration velocity has reached its threshold"

   window.speechSynthesis.speak(msg)

  }

    const msgh = () => {

      const msg1 = new SpeechSynthesisUtterance()

      msg1.text = "Warning warning humidity has reached its threshold"

      window.speechSynthesis.speak(msg1)

    }

    const msgc = () => {

      const msg2 = new SpeechSynthesisUtterance()

      msg2.text = "Warning warning Crack has been detected"

      window.speechSynthesis.speak(msg2)

    }

  const ih = Number(hum);

  const iv = Number(vib);

  const ick = Number(crks);

  const calc= () => {

    const ss = Number(confac) + Number(extfac);

    const pr = lifespan - ss;

    const prr = pr - Number(bage);

      setpredicted(prr);

  }

  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">Cracks :</strong>

          </Toast.Header>

          <Toast.Body>

          { ick > 150 ?  <div style={{display: "flex",backgroundColor: "red", borderRadius:"50%", width:"100px",height:"100px", marginLeft:50, marginTop:10}}>

            <p style={{margin: "auto"}}>Crack has been detected</p>{msgc()}

            </div> : <div style={{display: "flex",backgroundColor: "green", borderRadius:"50%", width:"100px",height:"100px", marginLeft:50, marginTop:10}}>

              <p style={{margin: "auto"}}>No Cracks</p>

           </div>}

      </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">vibration velocity:</strong>

            <small>{vib} m/s</small>

          </Toast.Header>

          <Toast.Body>

          {iv > .0508 ? <> <Alert variant='danger'> Warning!! warning!! vibration velocity has reached its threshold</Alert> {msgv()}</> : ""}

          <Sparklines data={vibarr} 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>

      <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 > 70 ? <><Alert variant='danger'>Warning!! warning!! humidity has reached its threshold</Alert>{msgh()}</>: ""}

          <Sparklines data={humarr} 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>

    </Row>

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

      <Row>

        <Col>

        <input placeholder='initial life span' onChange={(e) => setlifespan(e.target.value)} style={{width: 80}} />

        </Col>

      <Col>

      <input placeholder='age' onChange={(e) => setbage(e.target.value)} style={{width: 40}}  />

      </Col>

      </Row>

      <Row>

      <input placeholder='condition factor' value={confac} onChange={(e) => setconfac(e.target.value)}  />

      </Row>

      <Row>

      <input placeholder='external factor' onChange={(e) => setextfac(e.target.value)} disabled/>

      </Row>

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

    <p>remaining age:{predicted}</p>

    </div>

</>

  );

}

export default App;

**the code of the Arduino:**

//Libraries

#include <DHT.h>;

#include <ArduinoJson.h>

//Constants

#define DHTPIN 8

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

// Define Trig and Echo pin:

#define trigPin 2

#define echoPin 3

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

// Define variables:

long duration;

int distance;

//////////mpu

#include <MPU6050\_tockn.h>

#include <Wire.h>

#include <Arduino\_JSON.h>

MPU6050 mpu6050(Wire);

#define inA 4

#define inB 5

#define en 6

float mpu\_y;

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

//Variables

int chk;

int hum; //Stores humidity value

int temp; //Stores temperature value

void setup()

{

Serial.begin(9600);

//////////mpu

pinMode(inA,OUTPUT);

pinMode(inB,OUTPUT);

Wire.begin();

mpu6050.begin();

//mpu6050.calcGyroOffsets(true);

mpu6050.setGyroOffsets(0.59,-0.34,-2.02);

mpu6050.update();

mpu\_y = mpu6050.getAngleY();

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

dht.begin();

// Define inputs and outputs

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

delay(2000);

}

void loop()

{

//////////////////////mpu

mpu6050.update();

int val = mpu\_y-mpu6050.getAngleY();

int valx = mpu6050.getAngleX();

int valz= mpu6050.getAngleZ();

int pwm = constrain(map(abs(val),0,30,50,170),0,255);

if(val > 5){

digitalWrite(inA,HIGH);

digitalWrite(inB,LOW);

analogWrite(en,pwm);

}

else if(val < -5){

digitalWrite(inA,LOW);

digitalWrite(inB,HIGH);

analogWrite(en,pwm);

}

else{

digitalWrite(inA,LOW);

digitalWrite(inB,LOW);

analogWrite(en,0);

}

hum = dht.readHumidity();

temp= dht.readTemperature();

digitalWrite(trigPin, LOW);

delayMicroseconds(5);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);

distance = (duration/2) / 29.1;

Serial.print("Distance:");

Serial.println(distance);

Serial.print("vib:");

Serial.println(mpu6050.getAccZ()/9.8);

Serial.print("Humi:");

Serial.println(hum);

Serial.print("Temp:");

Serial.println(temp);

delay(2000); //Delay 2 sec.

}