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'

});

// declare data base variables

class Params extends Model {}

Params.init({

  Accz: DataTypes.INTEGER,

  Co2: 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);

  });\*/

});

//connect arduino

const port = new SerialPort({ path: 'COM14',baudRate: 9600,}, function (err) {

  if (err) {

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

  }

})

//connect esp

const portb = new SerialPort({ path: 'COM19',baudRate: 38400,}, 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')

})

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

  if (err) {

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

  }

  console.log('message b written')

})

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

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

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

    port.read();

  })

  portb.on('readable b', function () {

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

    portb.read();

  })

let t = '';

let co2 = '';

let ac = '';

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

  //})\*/

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

    parser.on('data',(data) => {

      //console.log(data);

    switch (true) {

        case data.startsWith("temperature"):

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

          t = data.slice(12);

          break;

        case data.startsWith("TVOC"):

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

          co2= data.slice(5);

          break;

        /\*case data.startsWith("Humi"):

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

          h = data.slice(5);

          break;    \*/

      }

     /\* try{

        io.emit("receive\_message", t, co2 );

       // Params.create({Temp: t, Co2:co2 });

      }catch(e){

        console.log(e);

      }\*/

      console.log(t);

      console.log(co2);

    })

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

    parserb.on('data',(data) => {

     // console.log(data);

    switch (true) {

        case data.startsWith("ACCZ"):

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

         ac = data.slice(5);

          break;

      }

      try{

       io.emit("receive\_message", ac, co2, t);

       // Params.create({accz: ac});

       Params.create({Temp: t, Co2:co2 , Accz: ac});

      }catch(e){

        console.log(e);

      }

      console.log(ac);

    })

  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())