|  |
| --- |
| **МИНОБРНАУКИ РОССИИ**  федеральное государственное бюджетное образовательное  учреждение высшего образования  **«Национальный исследовательский университет «МЭИ»** |

|  |  |
| --- | --- |
| **Институт** |  |
| **Кафедра** |  |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**(бакалаврская работа)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Направление** | | | |  | | | | | |
| (код и наименование) | | | | | | | | | |
| **Направленность (профиль)** | | | | | | | | |  |
|  | | | | | | | | | |
|  | | | | | | | | | |
| **Форма обучения** | | | | | | |  | | |
|  | | (очная/очно-заочная/заочная) | | | | | | | |
| **Тема:** | |  | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
| **Студент** | | |  | | | | | | |
| группа подпись фамилия и инициалы | | | | | | | | | |
| **Научный руководитель** | | | | | |  | | | |
| уч. степень должность подпись фамилия и инициалы | | | | | | | | | |
| **Консультант** | | | | | |  | | | |
| уч. степень должность подпись фамилия и инициалы | | | | | | | | | |
| **Консультант** | | | | | |  | | | |
| уч. степень должность подпись фамилия и инициалы | | | | | | | | | |
|  | | | | | | | | | |
| **«Работа допущена к защите»** | | | | | | | | | |
| **Зав. кафедрой** | | | | |  | | | | |
| уч. степень звание подпись фамилия и инициалы | | | | | | | | | |
| **Дата** |  | | | | | | |  | |
|  | | | | | | | |  | |
| **Москва, 2022** | | | | | | | | | |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc112016536)

[1 Обоснование температурных диапазонов 4](#_Toc112016537)

[2 Демонстрация работы 6](#_Toc112016538)

[ЗАКЛЮЧЕНИЕ 9](#_Toc112016539)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 10](#_Toc112016540)

[ПРИЛОЖЕНИЕ А 11](#_Toc112016541)

# ВВЕДЕНИЕ

Силовые маслонаполненные трансформаторы — это наиболее распространенный тип трансформаторов, применяемых для распределения электроэнергии. Это самый важный компонент, который можно найти в системах энергообеспечения в любой стране мира.

Современные мощные силовые трансформаторы имеют преимущественно масляное охлаждение. С введением в трансформатор минерального масла увеличивается пожарная опасность так как масло горит. Пары трансформаторного масла и смесь с воздухом воспламеняются под действием электрической дуги, искр и тому подобных факторов. Масло обладает высокими электроизоляционными свойствами. Благодаря своей малой вязкости и высокой текучести оно глубоко проникает в поры волокнистой изоляции (бумага, картон и др.) повышая её электрическую прочность. Одновременно, масло является хорошим переносчиком тепла и применяется для охлаждения трансформатора.

Целостность системы изолирования трансформатора, заполненного маслом, частично зависит и от состояния масла. Наиболее распространенной практикой, принятой в большинстве сетей снабжения электроэнергией во всем мире, является предоставление трансформаторам возможности охлаждаться естественным образом, когда изолирующая жидкость расширяется и сжимается в соответствие с нагрузкой.

В масляных трансформаторах при перегреве обмоток и магнитопровода или образовании электрических дуг изоляционное масло может воспламениться, причем вероятность воспламенения возрастает при перегреве масла, например, вследствие перегрузки трансформатора. Так как в масле постоянно содержится некоторое количество растворенного кислорода то при работе трансформатора всегда имеются условия для воспламенения масла. Газы образующиеся при термическом распаде масла могут прорвать стенку или крышку бака и тогда горение получает полное развитие превращаясь в пожар.

Длительное или значительное превышение температуры масла у верхних слоев маслонаполненного трансформатора свидетельствует о внутреннем повреждении трансформатора.

Задачи:

* Определение температурных диапазонов эксплуатации маслонаполненного трансформатора;
* Создание принципиальной схемы соединения элементов;
* Разработка алгоритма работы устройства и его программная реализация;
* Верификация работы устройства.

1. Обоснование температурных диапазонов

В работе рассматривается защита маслонаполненного силового трансформатора с естественной циркуляцией масла.

Для трансформаторов с естественным масляным охлаждением ( М) и принудительной циркуляцией воздуха Д при работе устанавливается небольшой перепад температуры между обмоткой и верхними слоями масла ввиду небольшой скорости масла в баке трансформатора, и для них допустимая температура масла составляет при номинальной нагрузке 95 [1].

Нормальный режим работы трансформаторного масла лежит в диапазоне 30-55 . При перегрузках трансформатора происходит нагрев масла. Температура свыше 70 ведет к ускоренному старению изоляции трансформатора и повышает вероятность возникновения внутреннего повреждения. Основываясь на [2] предложено принять, что при превышении температуры на 30% относительно предельно-нормальной (55 ) т.е. при температуре масла в верхних слоях порядка 70 . Допустимая длительность перегрузки составляет 120 минут.

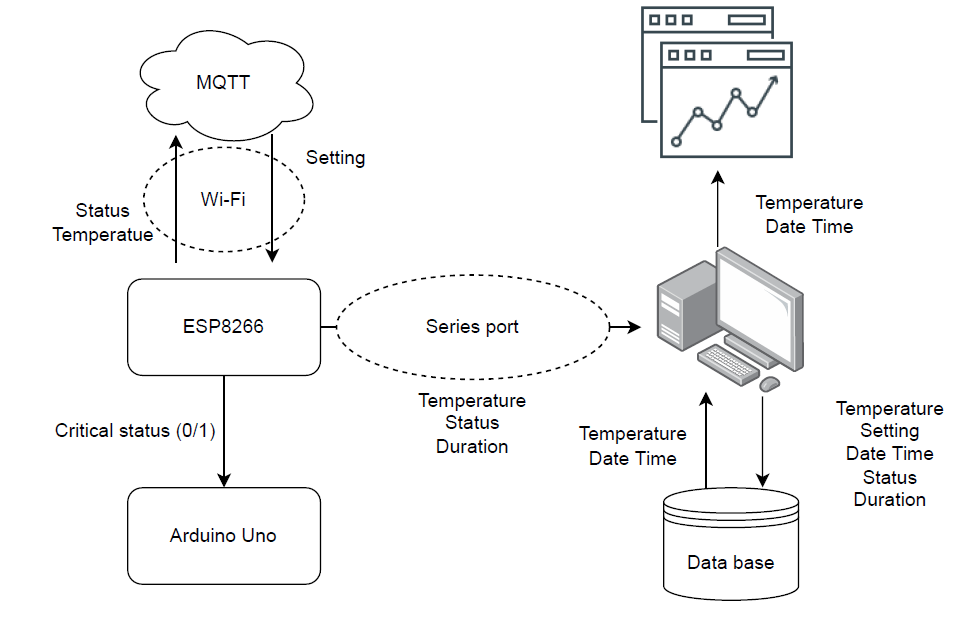
Таким образом, при превышении температуры масла значения в 70 существование такого режима допускается в течение 120 минут. Превышение температур масла значения в 95 недопустимо. При такой температуре должны приниматься меры по отключению установки.

1. Демонстрация работы

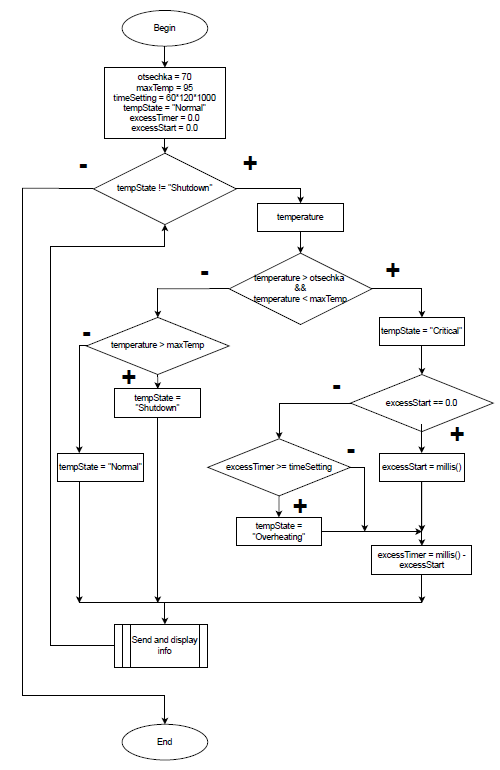
Разработанный комплекс включает следующие блоки:

1. Микроконтроллер ESP8266 выполняющий следующие функции:
   1. Получение данных о температуре масла с датчика …;
   2. Проверка температуры масла по условиям длительно допустимой и предельно допустимой температуры;
   3. Отображение информации о состоянии масла, текущей температуры, текущей уставки по длительно допустимой температуре, длительности превышения температуры;
   4. Отправка сигнала о превышении температуры на микроконтроллер Arduino Uno;
   5. Отправка данных по температуре, текущей уставке, состоянии масла по последовательному порту на устройство приема (ноутбук);
   6. Связь с сервером MQTT через Wi-Fi сеть для отправки данных о состоянии масла и текущей температуры на сервер, а также, для получения величины текущей уставки;
2. Микроконтроллер Arduino Uno:
   1. Получение состояния масла (0 – температура не превышена, 1 – превышена допустимо длительная температура);
   2. Включение светодиодной ленты для сигнализации о превышении температуры;
3. Сервер на устройстве прием (ноутбук), осуществляющий запись данных о состоянии масла в базу данных, а также отображение графика текущей температуры масла.

Принципиальная схема взаимодействия устройств приведена на рисунке 1.

**Рис. 1 – принципиальная схема взаимодействия устройств.**

Основные функции по защите трансформатора от перегрева масла осуществляет микроконтроллер ESP8266 (Приложение А). Данная плата осуществляет проверку допустимости температуры масла, оценку продолжительности перегрева и сигнализацию о температуре масла для проведения дальнейших мероприятий для выявления и ликвидации потенциальных повреждений оборудования. Алгоритм контроля температуры микроконтроллером ESP8266 приведен на рисунке 2:



**Рис. 2 – принципиальная схема взаимодействия устройств.**

Кратко опишем работу остальных элементов схемы рисунка 1:

Связь с MQTT сервером организованна с помощью глобальной сети интернет. С помощью библиотеки ESP8266WiFi организуется подключение к заранее определенной точке доступа, далее создается клиент с использованием библиотеки PubSubClient. Данный клиент подключается к MQTT брокеру и осуществляет:

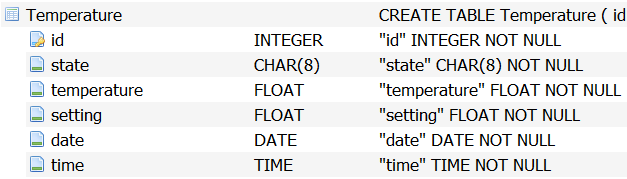
* Передачу текущего уровня температуры масла (“Normal”, “Critical”, “Shutdown”, “Overheating”) а также отправляет снимаемую с датчика температуру;
* Получение установленной пользователем длительно допустимой температуры нагрева масла.

Связь с микроконтроллером ArduinoUno осуществляется посредствам отправки логической единицы на аналоговой вход. Если температура не превышает длительно допустимой, на аналоговый pin Arduino подается логический 0. Иначе, подается логическая 1.

Микроконтроллер ArduinoUno осуществляет запуск сигнализации о перегреве масла трансформатора. При получении от ESP8266 логической единицы запускается скрипт сигнализации. Программный код Arduino можно посмотреть в Приложении Б.

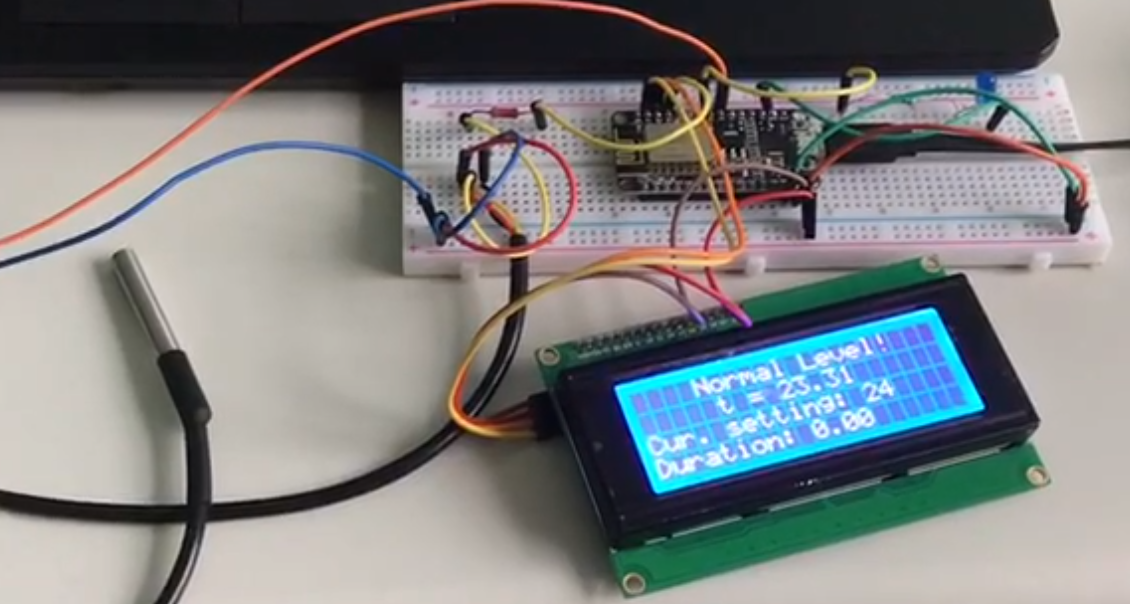
Связь с компьютером осуществляется посредствам последовательного порта. ESP8266 отправляет данные по состоянию масла, температуре масла, текущей уставке длительно допустимого превышения температуры. Посредствам Python происходит считывание данных по последовательному порту. Далее происходит обработка и запись данных в базу данных (Приложение В).

База данных SQLite содержит поля, приведенные на рисунке 3:

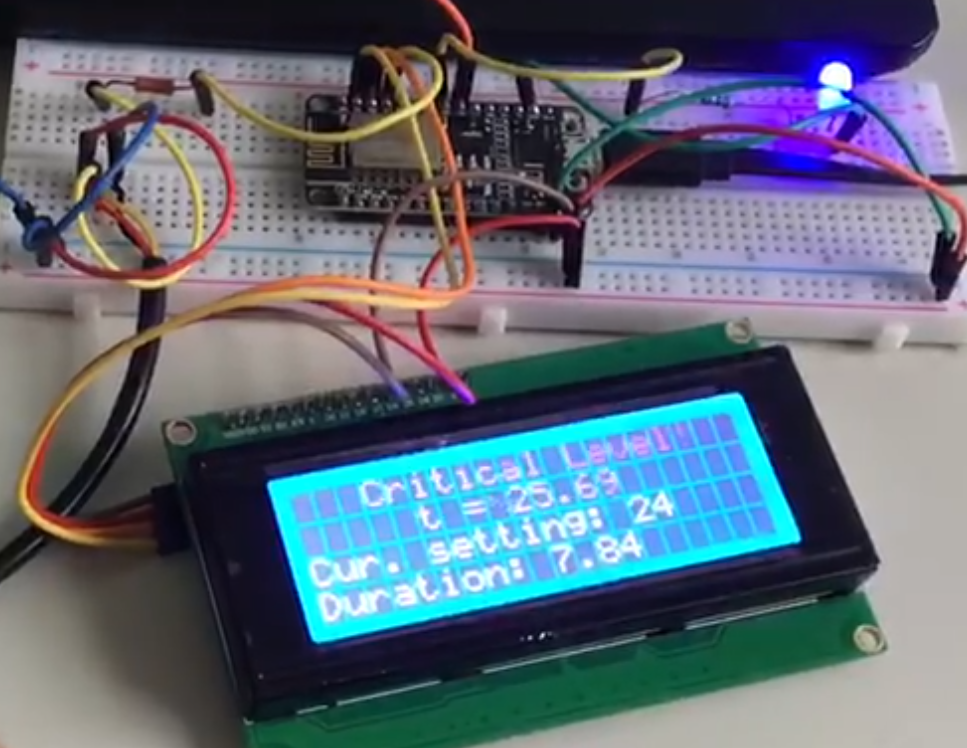
**Рис. 3 – содержание базы данных SQLite**

Построение графика на странице html происходит посредствам серверного фреймворка Python - Flask и фреймворка Java Script - Ajax. Код программы и html приведены в приложении Г.

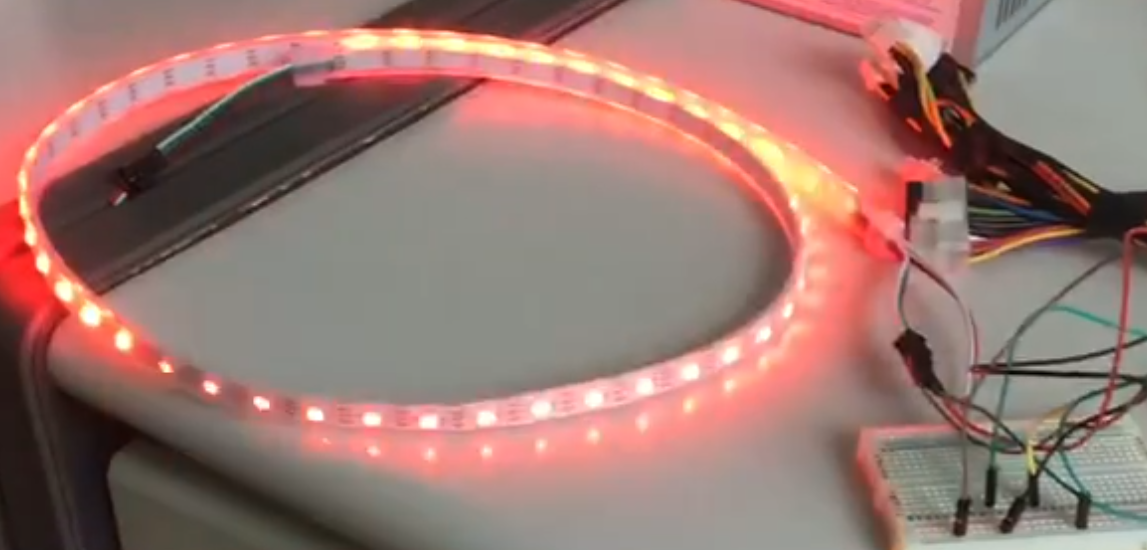
Фотографии работы схемы представленных на рисунках 4-8:



**Рис. 4 – работа датчика и дисплея в нормальном режиме**



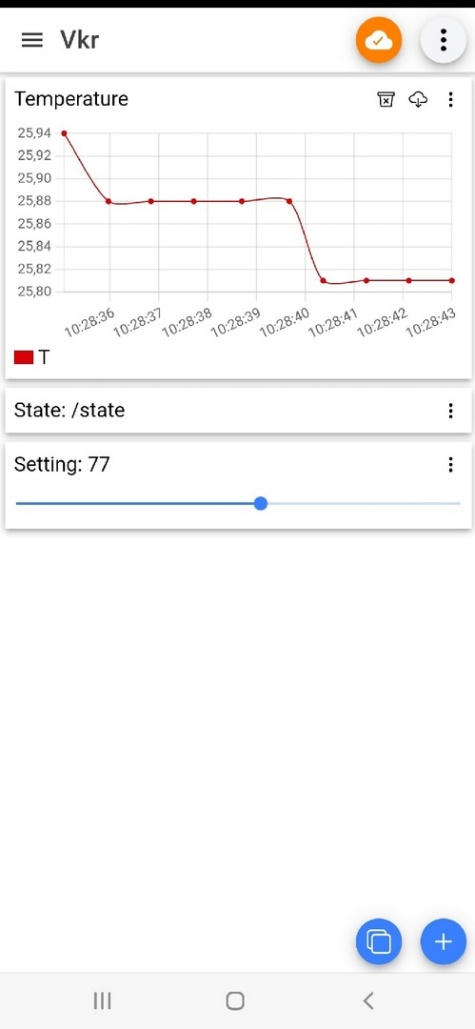
**Рис. 5 – работа датчика, дисплея и светодиода при превышении допустимой температуры**



**Рис. 6 – светосигнализация с помощью светодиодной ленты и ArduinoUno**



**Рис. 7 – построенние данных температуры в реальном времени на собственном сервере**



**Рис. 8 – отображение данных температуры в реальном времени на сервере MQTT и регулирование уставки**

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был выбран объект для контроля температуры. На основании условий безопасной работы и экономической эффективности были выбраны температурные диапазоны его работы. Для контроля температуры объекта был разработан алгоритм работы микроконтроллера в составе платы «Arduino Uno» и выполнена его программная реализация на языке «C++». Была спроектированная принципиальная схема соединения элементов. Верификация схемы и алгоритма производилась с помощью функционала сайта tinkercad.com [2]. Она показала, что разработанный прибор успешно выполняет все возложенные на него функции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Правила технической эксплуатации потребителей (ПТЭЭП) п.2.1.23.
2. Правила технической эксплуатации потребителей (ПТЭЭП) п.2.1.21.

ПРИЛОЖЕНИЕ А

КОД ESP8266

#include <OneWire.h>

#include <DallasTemperature.h>

#include <LiquidCrystal\_I2C.h>                  // I2C LCD

#include <Wire.h>                 // I2C

#include <ESP8266WiFi.h>  // Встроена в библиотеку ESP8266

#include <PubSubClient.h>

#define ONE\_WIRE\_BUS 0 // Пин подключения OneWire шины, 0 (D3)

#define diodPin 14 // пин выхода D5 для диода

#define connectionPin 12 // D6 pin ардуино

// create LCD object

LiquidCrystal\_I2C LCD (0x27 , 20, 4);

OneWire oneWire(ONE\_WIRE\_BUS); // Подключаем библиотеку OneWire

DallasTemperature sensors(&oneWire); // Подключаем библиотеку DallasTemperature

DeviceAddress adreses {0x28, 0xFF, 0x94, 0x32, 0x60, 0x17, 0x03, 0x6A};

float temperature;

float excessTimer = 0.0, excessStart = 0.0;

int otsechka = 70;                // Базовая уставка по временно допустимому перегреву

int maxTemp = 95;                 // Предельная температура

float timeSetting = 60\*120\*1000;  // Уставка по времени в миллисекундах

char\* tempState = "Normal";

const char\* ssid = "Rush";

const char\* password =  "09012000";

const char\* mqttServer = "193.233.67.51";

const int mqttPort = 1883;

const char\* graph = "/graph";

const char\* state = "/state";

const char\* setting = "/setting";

WiFiClient espClient;

PubSubClient client(espClient);

class Led{

  public:

  int pin;

  bool state;

  Led(int initPin, bool initState = false){

    pinMode(initPin, OUTPUT);

    digitalWrite(initPin, initState);

    pin = initPin;

    state = initState;

  }

  void on(){

    if (state){

      return;

    }

    state = true;

    digitalWrite(pin, state);

  }

  void off(){

    if (!state){

      return;

    }

    state = false;

    digitalWrite(pin, state);

  }

};

void setup(void)

{

  pinMode(connectionPin, OUTPUT);

  Serial.begin(115200);     // Задаем скорость соединения с последовательным портом

  sensors.begin();          // Инициализируем датчики

                            // Инициализируем дисплей

  LCD.init();

  LCD.backlight();

  Led led(diodPin);

  WiFi.begin(ssid, password);      // Подключаемся к wi-fi

  Serial.println('Connecting: ');

  while (WiFi.status() != WL\_CONNECTED)

  {

    delay(500);

    Serial.print(".");

  }

  Serial.println();

  Serial.print("Connected, IP address: ");

  Serial.println(WiFi.localIP());

  client.setServer(mqttServer, mqttPort);

  client.setCallback(callback);

  while (!client.connected()) {

    Serial.println("Connecting to MQTT...");

    String client\_id = "ESP8266Client-";

    client\_id += String(WiFi.macAddress());

    if (client.connect(client\_id.c\_str())) {

      Serial.println("connected");

    }

    else {

      Serial.print("failed with state ");

      Serial.print(client.state());

      Serial.println();

      delay(2000);

    }

  }

  client.subscribe(setting);

}

void callback(char\* topic, byte\* payload, unsigned int length) {

  char settingValue[length];

  for (int i = 0; i < length; i++) {

    settingValue[i] = (char)payload[i];

  }

  otsechka = atof(settingValue); // Поменяли уставку

}

void blink\_display(LiquidCrystal\_I2C lcd, int startTime)

{

  lcd.noBacklight();

  delay(500);

  lcd.backlight();

  delay(500);

  lcd.noBacklight();

  delay(500);

  lcd.backlight();

}

void loop(void)

{

  client.loop();

  if (tempState != "Shutdown"){

    sensors.requestTemperatures();

    temperature = sensors.getTempC(adreses);

    LCD.setCursor(5, 1);

    LCD.print("t = ");

    LCD.print(temperature);

    Serial.print(temperature); // Выводим температуру с датчика

    Serial.print(' ');

    char t[5];

    dtostrf(temperature, 5, 2, t);  // Создали переменную для отправки данных на график

    if (temperature > otsechka && temperature < maxTemp)

    {

      tempState = "Critical";

      led.on();

      if (excessStart == 0.0)   // Запускаем таймер

      {

        excessStart = millis();

        blink\_display(LCD, excessStart);

        digitalWrite(connectionPin, HIGH);

      }

      else if(excessTimer >= timeSetting)

      {

        tempState = "Overheating";

      }

      excessTimer = millis() - excessStart;

      client.publish(graph, t);

      client.publish(state, tempState);

      LCD.setCursor(5, 1);

      LCD.print("t = ");

      LCD.print(temperature);

      delay(100);

      LCD.clear();

    }

    else if (temperature >= maxTemp){

      tempState = "Shutdown";

    }

    else

    {

      tempState = "Normal";

      led.off();

      digitalWrite(connectionPin, LOW);

      excessStart = 0.0;       // Сброс таймера

      excessTimer = 0.0;

      client.publish(graph, t);

      client.publish(state, tempState);

      delay(100);

    }

    Serial.print(tempState);       // Выводим состояние

    Serial.print(' ');

    Serial.println(otsechka);  // Выводим уставку

    LCD.setCursor(3, 0);      // Выводим данные на дисплей

    LCD.print(tempState);

    LCD.setCursor(0, 2);

    LCD.print("Cur. setting: ");

    LCD.print(otsechka);

    LCD.setCursor(0, 3);

    LCD.print("Duration: ");

    LCD.print(excessTimer / 1000);

  }

}

ПРИЛОЖЕНИЕ Б

КОД ArduinoUno

#define NUM\_LEDS 60

#include "FastLED.h"

#define PIN 14

#define connectionPin 19

CRGB leds[NUM\_LEDS];

byte counter;

void setup() {

  Serial.begin(9600);

  FastLED.addLeds<WS2811, PIN, GRB>(leds, NUM\_LEDS).setCorrection( TypicalLEDStrip );

  FastLED.setBrightness(200);

  pinMode(13, OUTPUT);

  pinMode(connectionPin, INPUT);

}

void blink() {

  for (int i = 0; i < NUM\_LEDS; i++ ) {         // от 0 до первой трети

    leds[i].setRGB(counter + i\*5, 0, 0);

  }

  counter += 3;        // counter меняется от 0 до 255 (тип данных byte)

  FastLED.show();

  delay(5);         // скорость движения радуги

  FastLED.clear();

  FastLED.showColor((0, 0, 0));

}

void loop() {

  if (digitalRead(connectionPin)){

    for (int j=0; j < 50; j++){blink();}

  }

}

ПРИЛОЖЕНИЕ В

КОД ДЛЯ ОБРАБОТКИ ДАННЫХ ПОСЛЕДОВАТЕЛЬНОГО ПОРТА И ЗАПИСИ В БАЗУ ДАННЫХ

**Код модуля db.py, осуществляющего взаимодействие с базой данных:**

import sqlite3 as sql  
import datetime  
  
  
def commit(db: str, query: str) -> None:  
 """Commits query to db  
 Args:  
 db (str): data base name  
 query (str): SQL commands  
 """  
 try:  
 conn = sql.connect(db)  
 cursor = conn.cursor()  
 cursor.execute(query)  
 conn.commit()  
 cursor.close()  
  
 except sql.Error as error:  
 print("[-] Error while connecting to sqlite", error)  
 finally:  
 if conn:  
 conn.close()  
  
  
def fetch\_data(db: str, query: str):  
 try:  
 conn = sql.connect(db)  
 cursor = conn.cursor()  
 cursor.execute(query)  
 data = cursor.fetchall()  
 cursor.close()  
 return data  
 except sql.Error as error:  
 print("[-] Error while connecting to sqlite", error)  
 finally:  
 if conn:  
 conn.close()  
  
  
def create\_db(db: str):  
 query = '''CREATE TABLE IF NOT EXISTS Temperature (  
 id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
 state CHAR(8) NOT NULL,  
 temperature FLOAT NOT NULL,  
 setting FLOAT NOT NULL,  
 date DATE NOT NULL,  
 time TIME NOT NULL);'''  
  
 commit(db, query)  
  
  
def insert\_data(db: str, temperature: float, setting: float, state: str):  
 cur\_date = datetime.date.today()  
 cur\_time = datetime.datetime.now().time()  
  
 values = f'\'{state}\', {temperature}, {setting}, \'{cur\_date}\', \'{cur\_time}\''  
 query = f'INSERT INTO {db}(state, temperature, setting, date, time) VALUES({values})'  
 commit(db, query)  
  
  
def prepare\_query(text: str):  
 text = text.strip()  
 text\_list = text.split()  
 return text\_list

**Код модуля serial\_connection.py, осуществляющего считывание с последовательного порта:**

from PySide6.QtSerialPort import QSerialPortInfo  
from serial import Serial  
from time import time  
  
MICROBIT\_PID = 60000  
MICROBIT\_VID = 4292  
  
def open\_port(port: QSerialPortInfo) -> None:  
 serial = Serial(port.portName(), 115200)  
 if serial.is\_open:  
 print(f'Serial port {port.portName()} is opened!')  
 return serial  
  
def read\_serial\_port(serial: Serial) -> float:  
 serial.flush()  
 msg = serial.readline()  
 text = str(msg, 'utf-8')  
 return text  
  
def get\_ports() -> list:  
 ports = QSerialPortInfo().availablePorts()  
 ports\_list = [port for port in ports]  
 return ports\_list  
  
def get\_active\_port(ports: list) -> QSerialPortInfo:  
 for port in ports:  
 pid = port.productIdentifier()  
 vid = port.vendorIdentifier()  
 if pid == MICROBIT\_PID and vid == MICROBIT\_VID:  
 return port  
 return None  
  
  
def connect\_serial():  
 ports = get\_ports()  
 active\_port = get\_active\_port(ports)  
 serial = open\_port(active\_port)  
 start = time()  
 while time() - start < 5:  
 read\_serial\_port(serial)  
 return serial

**Код модуля fill\_sql.py, осуществляющего запись в базу данных:**

from db import \*  
from serial\_connection import \*  
  
def main():  
 db\_name = 'Temperature'  
 create\_db(db\_name)  
  
 serial = connect\_serial()  
 try:  
 while True:  
 text = read\_serial\_port(serial)  
 text\_list = prepare\_query(text)  
 temperature, state, setting = text\_list  
 insert\_data(db\_name, temperature, setting, state)  
 except:  
 print('Error occurred!')  
 finally:  
 print('Work is stopped.')

ПРИЛОЖЕНИЕ Г

КОД СЕРВЕРА

**Запуск сервера и наполнение графика данными:**

import json  
import time  
  
from flask import Flask, Response, render\_template, stream\_with\_context  
from db.db import fetch\_data  
  
  
app = Flask(\_\_name\_\_)  
  
  
@app.route('/')  
def index():  
 return render\_template("index.html")  
  
  
@app.route('/chart-data')  
def chart\_data():  
 query = """SELECT temperature, time FROM Temperature Order by date DESC, time DESC LIMIT 1;"""  
  
 def load\_data():  
 while True:  
 data = fetch\_data('db/Temperature', query)  
 value = data[0][0]  
 t = data[0][1]  
  
 json\_data = json.dumps({'time': t,  
 'value': value})  
 yield f'data:{json\_data}\n\n'  
 time.sleep(0.1)  
  
 response = Response(stream\_with\_context(load\_data()), mimetype="text/event-stream")  
 response.headers["Cache-Control"] = "no-cache"  
 response.headers["X-Accel-Buffering"] = "no"  
 return response  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=True, threaded=True)

**Содержание html страницы:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Creating Real-Time Charts with Flask</title>

    <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">

    <link href="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.min.css" rel="stylesheet">

</head>

<body>

<div class="container">

    <div class="row">

        <div class="col-12">

            <div class="card">

                <div class="card-body">

                    <canvas id="canvas"></canvas>

                </div>

            </div>

        </div>

    </div>

</div>

<!--suppress JSUnresolvedLibraryURL -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>

<!--suppress JSUnresolvedLibraryURL -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.3.1/js/bootstrap.min.js"></script>

<!--suppress JSUnresolvedLibraryURL -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.min.js"></script>

<script>

    $(document).ready(function () {

        const config = {

            type: 'line',

            data: {

                labels: [],

                datasets: [{

                    label: "Температура",

                    backgroundColor: 'rgb(255, 99, 132)',

                    borderColor: 'rgb(255, 99, 132)',

                    data: [],

                    fill: false,

                }],

            },

            options: {

                responsive: true,

                title: {

                    display: true,

                    text: 'Данные по температуре'

                },

                tooltips: {

                    mode: 'index',

                    intersect: true,

                },

                hover: {

                    mode: 'nearest',

                    intersect: false

                },

                scales: {

                    xAxes: [{

                        display: true,

                        scaleLabel: {

                            display: true,

                            labelString: 'Time'

                        }

                    }],

                    yAxes: [{

                        display: true,

                        scaleLabel: {

                            display: true,

                            labelString: 'Temperature'

                        }

                    }]

                }

            }

        };

        const context = document.getElementById('canvas').getContext('2d');

        const lineChart = new Chart(context, config);

        const source = new EventSource("/chart-data");

        source.onmessage = function (event) {

            const data = JSON.parse(event.data);

            if (config.data.labels.length === 100) {

                config.data.labels.shift();

                config.data.datasets[0].data.shift();

            }

            config.data.labels.push(data.time);

            config.data.datasets[0].data.push(data.value);

            lineChart.update();

        }

    });

</script>

</body>

</html>