

Dokumentacja Projektu

„Asystent kontroli ciepła”

Projektowanie i programowanie systemów
mikroprocesorowych

Tomasz Słapiński

Rzeszów 2024r.

Spis treści

1. Wprowadzenie	2
2. Budowa	3
2.1. Opis budowy	3
2.2. Wyszczególnione elementy zestawu	3
2.3 Schemat	4
3. Kod oraz zasada jego działania	4
3.1. Arduino	5
3.1.1. Używane biblioteki	5
3.1.2. Opis kodu	5
3.2. ESP	16
3.2.1. Używane biblioteki	16
3.2.2. Opis kodu	16
4. Zdjęcia projektu	20
5. Podsumowanie	33

1. Wprowadzenie

Projekt "Asystent Kontroli Ciepła" ma na celu stworzenie systemu do zarządzania temperaturą w domu lub biurze. Z jego pomocą przestaniemy się martwić uciążliwością pamiętania o podkładaniu do kotła. Dzięki temu zaoszczędzimy czas i pojemność mózgu na poważniejsze sprawy.

Zasada działania systemu jest prosta i opiera się na metodyce działania kotłów gazowych. Temperatura pobierana jest z powietrza i na jej podstawie oraz na podstawie ostatnich danych o podłożeniu, system oblicza następny czas podłożenia i nas o tym informuje.

Informacje, które system wysyła użytkownikom to nic innego jak powiadomienia o wzrostach oraz spadkach temperatury. Jednak to nie wszystko. Projekt dba także o sytuacje kiedy inny domownik wyręczył nas i dzięki niemu nie musimy się martwić przez następne kilka godzin.

2. Budowa

2.1. Opis budowy

System zbudowany jest w oparciu o mikrokontroler Arduino UNO R3, który steruje czujnikiem temperatury, diodami, odbiornikiem podczerwieni, a także wyświetlaczem. Kolejnym elementem bez którego projekt nie znalazłby realnego zastosowania jest mikrokontroler ESP32, obsługujący WiFi oraz BlueTooth.

Mikrokontrolery współpracują ze sobą za pomocą połączenia szeregowego. Pin RX Arudino, połączony jest z pinem TX ESP, a pin RX ESP, połączony jest z pinem TX Arudino. W ten sposób otrzymujemy dwustronną komunikację między urządzeniami.

Można zatem powiedzieć, że zadaniem Arduino jest badanie warunków fizycznych oraz przedstawianie ich użytkownikowi, który jest blisko urządzenia – odpowiadają za to diody oraz wyświetlacz. Natomiast ESP dba o to, żeby użytkownik, który jest poza domem lub biurem był o wszystkim informowany.

2.2. Wyszczególnione elementy zestawu

Mikrokontroler Arduino UNO R3 – odpowiada za sterowanie czujnikiem temperatury, diodami, odbiornikiem podczerwieni, a także wyświetlaczem. Wysyła komunikaty do ESP i odbiera od niego odpowiedzi.

Mikrokontroler ESP32 – Odbiera komunikaty od Arduino UNO R3, reaguje na nie, a następnie wysyła odpowiedzi.

Czujnik moduł temperatury i wilgotności DHT11 – odpowiada za pomiary temperatury oraz wilgotności. W przyszłym rozwoju planowane jest przejście na jego nowszy odpowiednik DHT22, który jest zdecydowanie bardziej precyzyjny.

Niebieska dioda LED – odpowiada za wskazywanie odczytów temperatury poniżej 21 stopni Celsjusza. Wówczas jest zimno i trzeba należy podłożyć.

Czerwona dioda LED – odpowiada za wskazywanie temperatury umiarkowanej, czyli temperatury powyżej 21 stopni Celsjusza.

Dwa rezystory 1k Ω – odpowiadają za zmniejszenie napięcia na diodach, tak, aby się one nie spaliły.

Wyświetlacz LCD 2x16 znaków niebieski – odpowiada za wyświetlanie informacji użytkownikowi. Z uwagi na swój mały rozmiar – w sumie 32 pola na znaki, w systemie mamy możliwość sterowania monitorami, poprzez naciskanie strzałek na pilocie. Na każdym monitorze znajdują się inne ciekawe informacje.

Potencjometr – odpowiada za dostosowanie napięcia dostarczanego wyświetlaczowi. Kręcząc nim dostosowujemy nasilenie pól na wyświetlaczu. Jest to szeroko polecana taktyka, z której postanowiłem skorzystać.

Odbiornik podczerwieni – odpowiada za reakcję na przyciskanie guzików na pilocie. Za pomocą tego urządzenia możemy komunikować się z arduino – przełączać ekran, wysyłać informacje o podłożeniu pod kocioł.

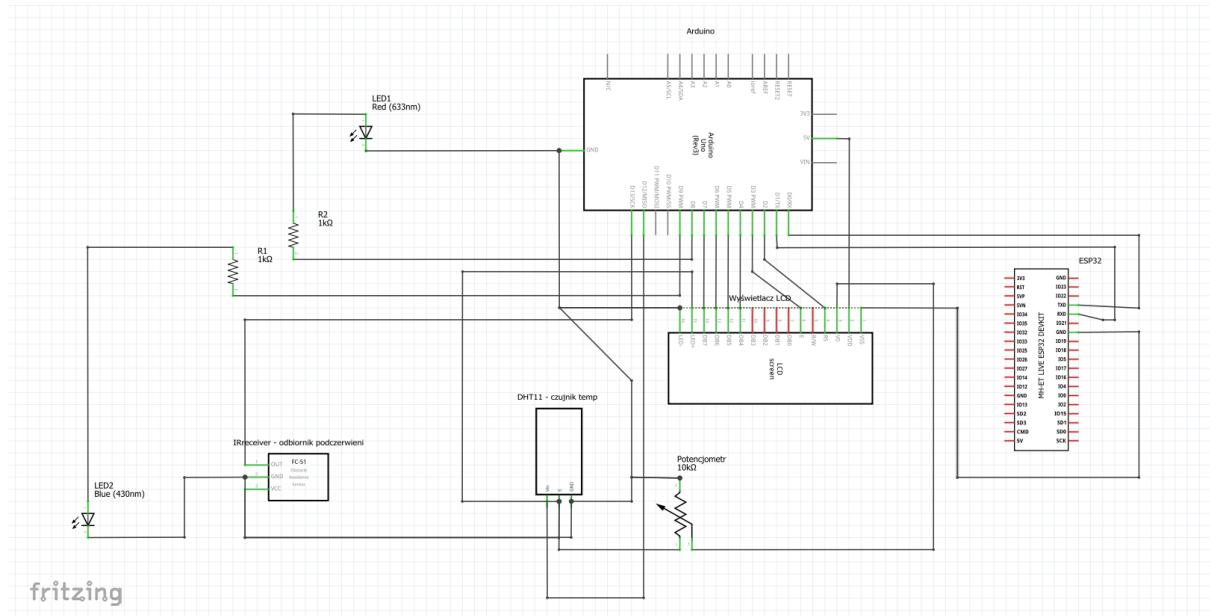
Pilot IR ELEGOO – służy do wysyłania komunikatów do wykonania dla Arduino.

Płyta stykowa 830 otworów – umożliwia tworzenie układów elektronicznych bez potrzeby lutowania

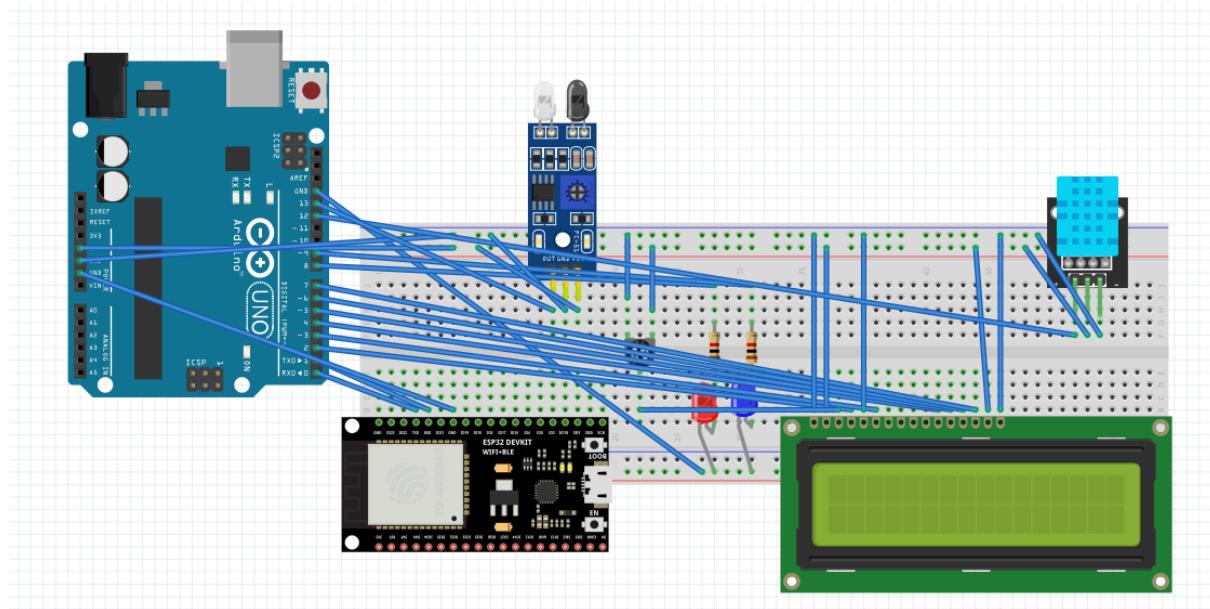
Przewody męsko-męskie oraz męsko-żeńskie – umożliwiają budowanie układów

2.3 Schemat

Schematic:



Breadboard:



3. Kod oraz zasada jego działania

Rozdział podzielony jest na część odpowiadającą zasadzie działania Arduino oraz część, odpowiadającą zasadzie działania ESP. Opis kodu obejmuje void setup oraz void loop.

Void setup() jest segmentem wykonywanym przy każdorazowym włączeniu urządzenia.

Void loop() jest segmentem wykonywanym nieustannie podczas gdy urządzenie jest włączone.

Autorskie funkcje opisane są w osobnych podrozdziałach.

3.1. Arduino

3.1.1. Używane biblioteki

LiquidCrystal.h – popularna biblioteka do obsługi alfanumerycznych wyświetlaczy ciekłokrystalicznych (LCD) w projektach opartych na platformie Arduino. Ta biblioteka dostarcza zestaw funkcji ułatwiających kontrolę nad wyświetlaczami LCD, umożliwiając programistom wygodne wyświetlanie tekstu i kontrolę nad interfejsem użytkownika.

IRremote.h – biblioteka, używana do obsługi odbiorników podczerwieni (IR) w projektach opartych na platformie Arduino. Ta biblioteka umożliwia odbieranie sygnałów z pilotów zdalnego sterowania, co jest przydatne w wielu projektach, takich jak zdalne sterowanie urządzeniami elektronicznymi, automatyką domową i innymi zastosowaniami związanymi z technologią podczerwieni.

DHT.h - Biblioteka DHT.h jest używana do obsługi cyfrowych czujników temperatury i wilgotności z serii DHT, takich jak DHT11, DHT21 (AM2301), DHT22 (AM2302), i innych. Te czujniki są popularne ze względu na swoją prostotę i dokładność w pomiarze temperatury i wilgotności. Biblioteka DHT.h ułatwia integrację tych czujników z platformą Arduino.

3.1.2. Opis kodu

3.1.2.1 Zmienne oraz stałe globalne

`#define DHTPIN 12`

Stałe określająca pin połączenia czujnika DHT z Arduino

`#define DHTTYPE DHT11`

Stałe określająca rodzaj czujnika temperatury DHT.

`DHT dht(DHTPIN, DHTTYPE);`

Instancja czujnika DHT11 podłączonego na pinie 12.

`LiquidCrystal lcd(2, 3, 4, 5, 6, 7);`

Instancja wyświetlacza z wymienionymi pinami.

`const int pinBlueDiod = 8;`

`const int pinRedDiod = 9;`

Piny dla diody czerwonej oraz niebieskiej.

`IRrecv irrecv(13);`

Instancja odbiornika podczerwieni, podłączonego na pinie 13.

`bool LCD_On = true;`

Zmienna określająca, czy wyświetlacz LCD jest włączony.

`int monitor = 0;`

Zmienna określająca numer aktualnie wyświetlonego monitora.

`int lastMonitorIndex = 3;`

Zmienna określająca ostatni numer monitora.

```
byte degreeSign = B11011111;
```

Zmienna służąca jako znak stopni w wyświetlaczu.

```
int temperature, maxTemperature = dht.readTemperature();
```

Zmienne określające temperaturę aktualną oraz maksymalną temperaturę.

```
int minTemperature = 100;
```

Zmienna z minimalną temperaturą.

```
int humidity = dht.readHumidity();
```

Zmienna określająca stopień wilgotności.

```
String lastUnderlayTime = "17:23";
```

```
String nextUnderlayTime = "19:23";
```

```
int timeToNextUnderlay = 22;
```

Zmienne kolejno z domyślnym ostatnim czasem podłożenia, następnym czasem podłożenia oraz czasem do następnego podłożenia.

```
unsigned int woodAmount = 0;
```

```
unsigned int coalAmount = 0;
```

Zmienne do ilości włożonego do pieca węgla oraz drewna.

```
unsigned long lastMeasureExecutionTime, lastIRExecutionTime,
```

```
lastChangeTimeToNextUnderlay = 0;
```

```
const unsigned long measuresInterval = 1000;
```

```
const unsigned long IRinterval = 500;
```

```
const unsigned long nextUnderlayInterval = 60000;
```

Zmienne używane do imitacji wielowątkowości przez Arduino, które jest jednowątkowe. Są to interwały, z których korzystają funkcje wywoływane w pętli. W ten sposób jesteśmy w stanie uniknąć dużo zbędnych obliczeń.

```
bool underlaying = false;
```

```
bool settingWood = false;
```

```
bool settingCoal = false;
```

Zmienne określające czy aktualnie podkładamy, czy aktualnie ustalamy ile drewienek dorożyliśmy, czy ustalamy liczbę węgla.

3.1.2.2 Void setup

```
lcd.begin(16, 2);
```

Informujemy, że nasz wyświetlacz ma 16 kolumn oraz 2 wiersze.

```
Serial.begin(9600);
```

Ta linijka inicjalizuje komunikację szeregową (Serial) na prędkość 9600 bitów na sekundę.

Umożliwia to komunikację między mikrokontrolerami, a także komputerem za pomocą portu szeregowego, co jest przydatne do debugowania i monitorowania danych.

```
dht.begin();
```

Zaczynamy korzystanie z czujnika temperatury.

```
IrReceiver.begin(13);
```

Zaczynamy korzystanie z czujnika podczerwieni.

```
showMonitor();
```

Autorska funkcja opisana, później, służąca do wpisania treści na obraz wyświetlacza lcd.

```
pinMode(pinBlueDiod, OUTPUT);
```

```
pinMode(pinRedDiod, OUTPUT);
```

Podłączenie pinów do sterowania diodami.

3.1.2.3 Void loop

```
if(underlaying != true){  
    receiveDataFromESP();  
    reactToIR();  
    changeTimeToNextUnderlay();  
    readTemperatureAndHumidity(temperature, humidity);  
}
```

Jeśli aktualnie nie podkładamy to:

- Staramy się odebrać dane od ESP
- Staramy się reagować na podczerwień
- Co określony interwał czasu dekrementujemy czas do następnego podłożenia
- Co określony czas pobieramy temperaturę i wilgotność.

3.1.2.4 Funkcje autorskie

```
void clearMonitor(){  
    lcd.clear();  
}
```

Funkcja po prostu czyści wyświetlacz lcd – kasuje wszystko co na nim jest.

```
void showMonitor(){  
    clearMonitor();  
    Serial.print("Zmieniono obraz na: ");  
    Serial.println(monitor);  
    switch(monitor){  
        case 0:  
            // first row  
            lcd.setCursor(0,0);  
            lcd.print("IN ");  
            lcd.print(temperature); //2  
            lcd.print((char)degreeSign); //1  
            lcd.print("C "); //1  
            lcd.print("OUT ");  
            lcd.print(humidity);  
            lcd.print("%");  
            // second row  
            lcd.setCursor(0,1);  
            lcd.print(timeToNextUnderlay);  
            lcd.print(" MINS LEFT");  
        break;  
        case 1:  
            lcd.setCursor(0,0);
```

```

        lcd.print("LAST: ");
        lcd.print(lastUnderlayTime);
        lcd.setCursor(0,1);
        lcd.print("NEXT: ");
        lcd.print(nextUnderlayTime);
    break;
case 2:
    lcd.setCursor(0,0);
    lcd.print(" MIN MAX");
    lcd.setCursor(0,1);

    lcd.print(" ");
    lcd.print(minTemperature);
    lcd.print((char)degreeSign); //1
    lcd.print("C "); //1
    lcd.print(" ");
    lcd.print(maxTemperature);
    lcd.print((char)degreeSign); //1
    lcd.print("C"); //1
break;
case 3:
    lcd.setCursor(0,0);
    lcd.print("OSTATNIE PODL.");
    lcd.setCursor(0,1);
    lcd.print("Drewna:");
    lcd.print(woodAmount);
    lcd.print(" Węgla:");
    lcd.print(coalAmount);
break;
}
}

```

Funkcja showMonitor() najpierw przygotowuje wyświetlacz LCD pod zapis, a następnie w zależności od zmiennej monitor wyświetla odpowiedni obraz.

Na pierwszym monitorze jest temperatura, wilgotność powietrza oraz ilość minut pozostałych do podłożenia pod kotłem.

Drugi monitor zawiera ostatnią godzinę podłożenia oraz następną godzinę podłożenia.

Trzeci monitor ma minimalną oraz maksymalną temperaturę.

Czwarty monitor zawiera informacje o ostatnim podłożeniu – ilość węgla oraz drewna.

```

void setMonitor(int m){
    if(monitor != m){
        monitor = m;
        showMonitor();
    }
}

```

Funkcja **setMonitor()** ustawia monitor na ten wybrany w parametrze oraz odświeża wyświetlacz.

```
void checkNewMinMaxTemperature(){
    if(temperature > maxTemperature){
        maxTemperature = temperature;
    }
    if(temperature < minTemperature){
        minTemperature = temperature;
    }
}
```

Funkcja **checkNewMinMaxTemperature()** służy do sprawdzania czy została osiągnięta nowa minimalna albo maksymalna temperatura.

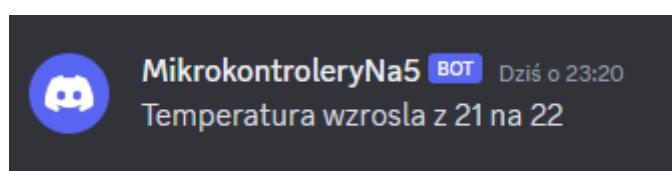
```
void readTemperatureAndHumidity(int temp, int hum){
    unsigned long currentMillis = millis();

    if (currentMillis - lastMeasureExecutionTime >= measuresInterval) {
        temperature = (int)dht.readTemperature();
        humidity = (int)dht.readHumidity();

        if(temp != temperature || hum != hum){
            SwitchDiod();
            if(temperature > temp) Serial.println("TO_ESP:Temperatura wzrosła z " +
String(temp) + " na " + String(temperature) + " stopnie");
            if(temperature < temp) Serial.println("TO_ESP:Temperatura zmalała z " +
String(temp) + " na " + String(temperature) + " stopnie");
            checkNewMinMaxTemperature();
            showMonitor();
            Serial.print("Nastąpiła zmiana odczytów: ");
            Serial.print(temperature);
            Serial.print(" *C ");
            Serial.print(humidity);
            Serial.println(" %");
        }

        lastMeasureExecutionTime = currentMillis;
    }
}
```

Funkcja **readTemperatureAndHumidity()** służy do pomiarów temperatury oraz wilgotności. Wykonuje się ona raz na sekundę. Jeśli wykryta nowa temperatura jest inna niż poprzednio zarejestrowana to wysyłamy informacje do ESP o tym jaką jest nowa temperatura, oraz czy względem poprzedniej zmalała, czy wzrosła. ESP następnie ma wysłać użytkownikowi wiadomość. Ostatecznie wygląda to tak.



```

void SwitchDiod(){
    if(temperature >= 21){
        digitalWrite(pinBlueDiod, LOW);
        digitalWrite(pinRedDiod, HIGH);
    }
    else {
        digitalWrite(pinRedDiod, LOW);
        digitalWrite(pinBlueDiod, HIGH);
        timeToNextUnderlay = 0;
        Serial.println("TO_ESP:Temperatura spadla na poziom 21*C\rWymagane
natychmiastowe podlozenie!");
    }
}

```

Funkcja SwitchDiod() jest wywoływana podczas checkNewMixMaxTemperature(). Odpowiada ona za odpowiednie gaszenie i zapalanie diód sygnalizujących natężenie temperatury. Ma także jedno dodatkowe zadanie. Jeśli temperatura spadnie na mniej niż 21 stopnie wówczas zeruje ona pozostały czas do podłożenia i wysyła komunikat do ESP do przesłania do użytkownika, że podłożenie jest natychmiast wymagane.

```

int CalcTimeForNextUnderlay(){
    return woodAmount * 20 + coalAmount * 60;
}

```

Ta prosta funkcja po prostu zwraca czas pozostały do podłożenia, w postaci ilości drewna * 20min + ilości węgla(łopatek) * 60min.

```

void receiveDataFromESP(){
    if (Serial.available() > 0) {
        char terminator = '\n'; // Znak końca linii
        String receivedText = Serial.readStringUntil(terminator);
        Serial.println(receivedText);
        if (receivedText.startsWith("TO_ARDUINO:")) {
            Serial.print("FROM_ESP: ");
            String receivedMessage = receivedText.substring(11);
            Serial.println(receivedMessage);
            if(receivedMessage.startsWith("Godzina podlozenia")){
                lastUnderlayTime = receivedMessage.substring(19,24);
                nextUnderlayTime = addTime(lastUnderlayTime,
CalcTimeForNextUnderlay());
                timeToNextUnderlay = CalcTimeForNextUnderlay();
                delay(1000);
                Serial.println("TO_ESP:Nastepne podlozenie przewidywane jest za:
"+String(timeToNextUnderlay) + "minut \rCzyli o godzinie:
"+String(nextUnderlayTime));
                showMonitor();
            }
        }
    }
}

```

Funkcja **receiveDataFromESP()** odpowiada za reagowanie na komunikaty od ESP do Arduino. Odczytujemy w niej zawartość komunikatu. Jeśli zaczyna się on od znaków „Godzina Podłożenia” wówczas ESP wysłało nam datę ostatniego podłożenia. Na jej podstawie obliczamy godzinę następnego podłożenia, a także pozostały do podłożenia czas. Na końcu odświeżamy monitor, tak, aby pokazać nowo uzyskane dane i wysyłamy do ESP dane do wysłania użytkownikowi.

```
void startUnderlay(){
    underlaying = true;
    settingWood = true;
    settingCoal = true;
    setWoodAmount();
    setCoalAmount();
    underlaying = false;
}
```

Funkcja **startUnderlay()** odpowiada za zaczęcie sekwencji podkładania rozpoczętej przez naciśnięcie guzika na pilocie.

```
void setWoodAmount(){
    clearMonitor();
    woodAmount = 0;
    while(settingWood == true){
        reactToIRDuringUnderlay();
        lcd.setCursor(0,0);
        lcd.print("Ile drewienek?");
        // second row
        lcd.setCursor(0,1);
        lcd.print(woodAmount);
    }
}
```

Pierwszym etapem podkładania jest ustalenie ile sztuk drewna dorzucamy. Wybieramy to klikając odpowiednie guzki na pilocie.

```
void setCoalAmount(){
    clearMonitor();
    coalAmount = 0;
    while(settingCoal == true){
        reactToIRDuringUnderlay();
        lcd.setCursor(0,0);
        lcd.print("Ile węgla?");
        // second row
        lcd.setCursor(0,1);
        lcd.print(coalAmount);
    }
    clearMonitor();
    lcd.setCursor(0,0);
    lcd.print("Pomyślana akcja!");
    // second row
    lcd.setCursor(0,1);
```

```

lcd.print("Drewna:");
lcd.print(woodAmount);
lcd.print(" Węgla:");
lcd.print(coalAmount);
delay(4000);
showMonitor();
}

```

Drugim etapem jest wybór ilości szufelek węgla. Po potwierdzeniu ilości otrzymuje na wyświetlaczu komunikat, że wszystko poszło OK.

Warto zwrócić uwagę, że w funkcjach do ustalania liczby drewna i węgla korzystamy z funkcji reactToIRDuringUnderlay(). Dzieje się tak ponieważ, musimy zapewnić inne działanie guzików normalnie a inne w trybie podkładania.

```

void reactToIRDuringUnderlay(){
    unsigned long currentMillis = millis();
    //Eliminacja podwojnych klików
    if (currentMillis - lastIRExecutionTime >= IRinterval) {
        if(IrReceiver.decode()){
            code = IrReceiver.decodedIRData.command;
            if(code != 0){
                Serial.print("Underlay + Nacisnieto kod= ");
                Serial.println(code);
                switch(code){
                    // 69 = czerwony wlacznik
                    case 69:
                        if(LCD_On){
                            lcd.noDisplay();
                            LCD_On = false;
                        }
                        else{
                            lcd.display();
                            LCD_On = true;
                        }
                        break;
                    // ST/REPT
                    case 13:
                    // PLAY/PAUSE
                    case 64:
                        if( underlaying == true && settingWood == true){
                            settingWood = false;
                        }
                        else if(underlaying == true && settingCoal == true){
                            settingCoal = false;
                        }
                        break;
                    // STRZALKA W PRAWO
                    case 67:
                }
            }
        }
    }
}

```

```

// STRZALKA W DOL
case 9:
    if(settingWood == true) woodAmount++;
    else coalAmount++;
    clearMonitor();
break;
// STRZALKA W LEWO
case 68:
// STRZALKA W DOL
case 7:
    if(settingWood == true){
        if(woodAmount > 0) woodAmount--;
    }
    else{
        if(coalAmount > 0) coalAmount--;
    }
    clearMonitor();
break;
// FUNC/STOP
case 71:
    if(settingWood == true){
        woodAmount = woodAmount / 10;
    }
    else{
        coalAmount = coalAmount / 10;
    }
    clearMonitor();
break;
}
lastIRExecutionTime = currentMillis;
}
}
}
IrReceiver.resume();
}

```

Komentarzami zaznaczone są guziki na pilocie odpowiadające danemu kodowi. Kiedy odbiornik dostanie jeden z wymienionych kodów do swojego bufora, a także odstęp między poprzednim otrzymaniem kodu przez odbiornik nie jest za mały, wówczas dzieje się akcja zdefiniowana w switchu.

```

void reactToIR(){
unsigned long currentMillis = millis();
//Eliminacja podwojnych klikow
if (currentMillis - lastIRExecutionTime >= IRinterval) {
if(IrReceiver.decode()){
code = IrReceiver.decodedIRData.command;
if(code != 0){

```

```

Serial.print("Nacisnieto kod= ");
Serial.println(code);
switch(code){
    // 7 = strzalka w gore
    case 7:
        if(monitor < lastMonitorIndex){
            monitor++;
        }
        else{
            monitor = 0;
        }
        showMonitor();
        break;
    // 9 = strzalka w dol
    case 9:
        if(monitor > 0){
            monitor--;
        }
        else{
            monitor = lastMonitorIndex;
        }
        showMonitor();
        break;
    // 69 = czerwony wlacznik
    case 69:
        if(LCD_On){
            lcd.noDisplay();
            LCD_On = false;
        }
        else{
            lcd.display();
            LCD_On = true;
        }
        break;
    // 64 = srodkowy play/stop
    case 64:
        lastIRExecutionTime = currentMillis;
        startUnderlay();
        Serial.println("TO_ESP:Podlozono pod kotlem\rIlosc drewna: " +
String(woodAmount) + " Ilosc węgla: " + String(coalAmount));
        break;
    case 12:
        setMonitor(0);
        break;
    case 24:
        setMonitor(1);
        break;
    case 94:
        setMonitor(2);
}

```

```

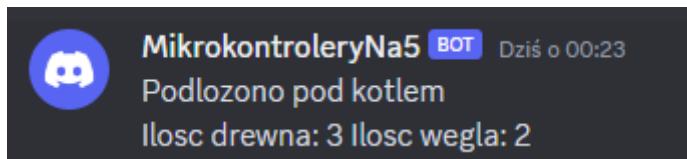
        break;
    }
    lastIRExecutionTime = currentMillis;
}
}
}
IrReceiver.resume();
}

```

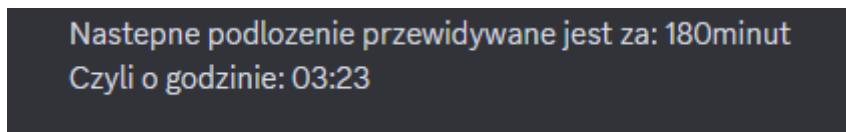
Funkcja **reactToIR()** działa dokładnie tak samo jak jej poprzednik z tą różnicą, że wykonuje się ona kiedy tryb podkładania nie jest aktywny. Warto też wspomnieć, że po wykonaniu się podłożenia to ona wysyła komunikat do ESP o tym, że podłożono pod kocioł.

Po podłożeniu za pomocą pilota każdy członek serwera dostaje powiadomienia:

1.



2.



```

String addTime(String originalTime, int minutesToAdd) {
    // Rozdziel godzinę i minutę
    int originalHour = originalTime.substring(0, 2).toInt();
    int originalMinute = originalTime.substring(3).toInt();

    // Dodaj godziny i minuty
    int totalMinutes = originalHour * 60 + originalMinute + minutesToAdd;

    int newHour = (totalMinutes / 60) % 24;
    int newMinute = totalMinutes % 60;

    // Sformatuj wynik w postaci "hh:mm"
    String newTime = String(newHour / 10) + String(newHour % 10) + ":" +
                    String(newMinute / 10) + String(newMinute % 10);

    return newTime;
}

```

Funkcja pomocnicza **addTime()** dodaje do godziny liczbę minut, uwzględniając w sobie szczególne przypadki.

```

void changeTimeToNextUnderlay(){
    unsigned long currentMillis = millis();
    //Eliminacja podwojnych klików
}

```

```

    if (currentMillis - lastChangeTimeToNextUnderlay >= nextUnderlayInterval &&
timeToNextUnderlay > 0) {
    timeToNextUnderlay--;
    Serial.print("Nowy czas to nastepnego podlozenia: ");
    Serial.println(timeToNextUnderlay);
    showMonitor();
    lastChangeTimeToNextUnderlay = currentMillis;
}
}

```

Funkcja **changeTimeToNextUnderlay()** odpowiada za minusowanie co minutę czasu pozostałego do podłożenia, a także do odświeżania tego czasu na monitorze.

3.2. ESP

3.2.1. Używane biblioteki

WiFi.h – Biblioteka WiFi.h jest biblioteką do obsługi funkcji związanych z bezprzewodową łącznością WiFi w projektach opartych na platformie Arduino ESP.

HTTPClient.h – biblioteka używana do nawiązywania połączeń http. Wysyłania żądań oraz otrzymywania odpowiedzi.

ArduinoJson.h – biblioteka używana do dwukierunkowej transformacji danych między obsługiwany przez ESP Stringiem a popularnym JSONem. Upraszczona format zapisu przesyłanych między urządzeniami danych.

3.2.2. Opis kodu

3.2.2.1 Zmienne oraz stałe globalne

`const char* ssid = "";`

Stała ssid musi zawierać nazwę sieci Wifi

`const char* password = "";`

Stała ssid musi zawierać nazwę sieci Wifi

`const char* discordServerAddress = "";`

Stała discordServerAddress zawiera adres discordowego webhooka

`const char* timeServerAddress =`

`"http://worldtimeapi.org/api/timezone/Europe/Warsaw";`

Stała timeServerAddress zawiera adres serwera do pobierania godziny

`const int serverPort = 80; // Port serwera HTTP`

Stała serverPort zawiera port do łączenia się z serwerami web

3.2.2.2. Void setup

Zaczniemy od tego co się wykonuje po każdorazowym włączeniu urządzenia.

`Serial.begin(9600);`

Ta linijka inicjalizuje komunikację szeregową (Serial) na prędkość 9600 bitów na sekundę. Umożliwia to komunikację między mikrokontrolerami, a także komputerem za pomocą portu szeregowego, co jest przydatne do debugowania i monitorowania danych.

`WiFi.begin(ssid, password);`

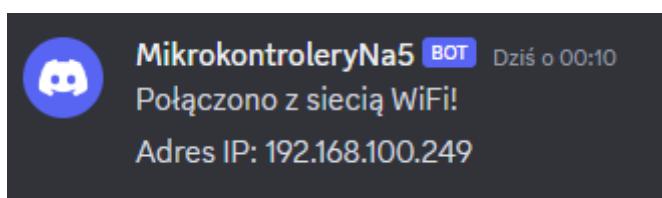
Rozpoczynamy próbę połączenia z siecią WiFi, używając nazwy sieci (SSID) i hasła podanych jako zmienne ssid i password. Wartości te muszą być wcześniej zdefiniowane w programie.

```
while (WiFi.status() != WL_CONNECTED) {  
    delay(250);  
    Serial.print("Trwa laczenie z Wifi\n");  
}
```

Pętla while sprawdza status połączenia WiFi. Czekamy, aż moduł ESP połączy się z siecią. W trakcie oczekiwania, co 250 milisekund, na konsolę szeregową wysyłane jest powiadomienie "Trwa laczenie z Wifi"

```
sendHttpPostRequest("Połączono z siecią WiFi!");
```

Wywołuje funkcję sendHttpPostRequest() z argumentem "Połączono z siecią WiFi!". Jest to autorska funkcja opisana w podrozdziale 3.2.2.3, odpowiadająca za przesłanie powiadomienia na discord z informacją o ustaleniu połączenia.



```
Serial.println("Połączono z siecią WiFi!");
```

Informacje te wysyłamy także na port szeregowy.

```
sendHttpPostRequest("Adres IP: " + WiFi.localIP().toString());
```

```
Serial.println("Adres IP: " + WiFi.localIP().toString());
```

Później wysyłamy na discord oraz port szeregowy przydzielony lokalny adres IP.

3.2.2.3. Void loop

```
receiveDataFromArduino();
```

To jedyna linia wykonywana nieustannie podczas działania ESP. Nasłuchuje ona danych otrzymanych od ESP i reaguje na nie. Opisana jest w podrozdziale 3.2.2.3

3.2.2.4 Autorskie funkcje

```
void sendHttpPostRequest(String content) {  
    // Utwórz obiekt klienta HTTP  
    HTTPClient http;  
    if (!WiFi.status() == WL_CONNECTED){  
        Serial.println("Nie ma polaczenia z WiFi - nie mozna wykonac  
sendHttpPostRequest()");  
    }  
    // Połącz z serwerem  
    if (http.begin(discordServerAddress)) {  
        Serial.println("Wykonuje sendHttpPostRequest()");  
  
        // Skonfiguruj nagłówki  
        http.addHeader("Content-Type", "application/json");  
  
        StaticJsonDocument<200> doc;  
        doc["content"] = content;
```

```

String postData;
serializeJson(doc, postData);

// Wyślij żądanie POST
int httpResponseCode = http.POST(postData);

// Odczytaj odpowiedź z serwera
if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
} else {
    Serial.print("Funkcja sendHttpPostRequest() zakończyła się błędem: ");
    Serial.println(httpResponseCode);
}
http.end();
} else {
    Serial.println("Błąd podczas wykonywania funkcji sendHttpPostRequest()");
}
}

```

Ta funkcja służy do wysłania requesta typu POST poprzez webhook na stworzony przeze mnie serwer discord. Funkcja dba o to, aby w przypadku jakichś niedogodności jak np. brak sygnału Wifi lub przy błędach ustanawiania połączenia i błędach samej odpowiedzi na żądanie, odpowiednie komunikaty były wysyłane na port szeregowy. Korzystam tutaj z biblioteki ArduinoJSON i zamieniam String podany w argumencie funkcji na odpowiedni JSON. Discord rozumie ten zapis i odbiór przez niego tej wiadomości równa się wysłaniu na telefon użytkownika powiadomienia oraz wiadomości o treści użytej w argumencie.

```

void receiveDataFromArduino(){
    if (Serial.available() > 0) {
        char terminator = '\n'; // Znak końca linii
        String receivedText = Serial.readStringUntil(terminator);
        if (receivedText.startsWith("TO_ESP:")) {
            String receivedMessage = receivedText.substring(7);
            Serial.print("FROM_ARDUINO: ");
            Serial.println(receivedMessage);

            sendHttpPostRequest(receivedMessage);
            if(receivedMessage.startsWith("Podłożono pod kotlem")){
                getTimeRequest();
            }
        }
    }
}

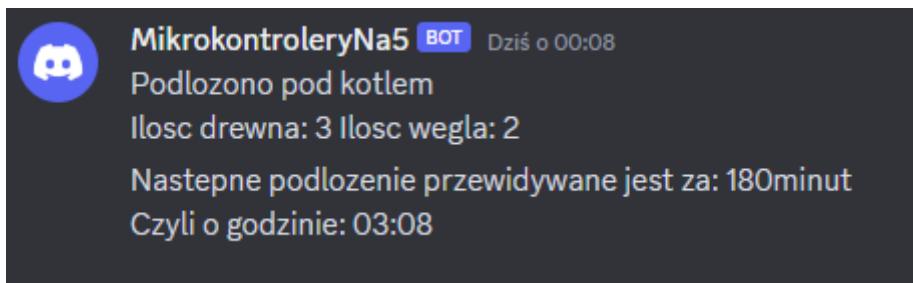
```

receiveDataFromArduino() to nieustannie wykonująca się funkcja. Zaczynamy w niej od sprawdzenia, czy jakieś dane są dostępne na porcie szeregowym. Jeśli tak, to odczytujemy

całą linię z bufora aż do znaku nowej linii. Później sprawdzamy czy owy ciąg zaczyna się od znaków „TO_ESP:”. Jeśli tak jest, wówczas wiemy, że na ten komunikat ESP musi zareagować.

Reakcja zaczyna się od wyciągnięcia reszty linii z otrzymanego ciągu znaków. Następnie ciąg ten wysyłamy jako powiadomienie discord, funkcją sendHttpPostRequest().

Jeśli otrzymany komunikat rozpoczyna się od „Podłożono pod kotlem”, wówczas korzystamy z funkcji getTimeRequest(). Działanie tej funkcji będzie opisane poniżej jednak warto przed omówieniem jej, napisać do czego ona jest potrzebna. Otóż aby zorientować się jaki jest aktualny czas na świecie to musimy skądś go otrzymać. Tak się składa, że do tego potrzebny jest specjalny moduł, którego nie posiadam. Dlatego dobrym rozwiązaniem jest po prostu skorzystanie z internetu i pobranie czasu za pomocą API.



```
void getTimeRequest(){
    // Utwórz obiekt klienta HTTP
    HTTPClient http;
    if (!WiFi.status() == WL_CONNECTED){
        Serial.println("Nie ma połączenia z WiFi - nie można wykonac
getTimeRequest()");
    }
    // Połącz z serwerem
    if (http.begin(timeServerAddress)) {
        Serial.println("Wykonuje getTimeRequest()");

        // Skonfiguruj nagłówki
        http.addHeader("Content-Type", "application/json");

        // Wyślij żądanie POST
        int httpResponseCode = http.GET();

        // Odczytaj odpowiedź z serwera
        if (httpResponseCode > 0) {
            Serial.print("HTTP Response code: ");
            Serial.println(httpResponseCode);

            String response = http.getString();
            Serial.print("Response: ");
            Serial.println(response);

            StaticJsonDocument<200> doc;
```

```

        deserializeJson(doc, response);
        String datetimeValue = doc["datetime"];
        datetimeValue = datetimeValue.substring(11, 19);

        Serial.print("TO_ARDUINO:Godzina podlozenia=");
        Serial.println(datetimeValue);
    } else {
        Serial.print("Funkcja getTimeRequest() zakonczyla sie bledem: ");
        Serial.println(httpResponseCode);
    }

    http.end();

} else {
    Serial.println("Blad podczas wykonywania funkcji getTimeRequest()");
}
}

```

Struktura funkcji **getTimeRequest()** jest podobna do funkcji omawianych poprzednio, z tą różnicą, że wysyła ona żądanie typu GET na adres

<http://worldtimeapi.org/api/timezone/Europe/Warsaw>

Z odpowiedzi serwera łuskamy czas w Polsce i wysyłamy zwrot do Arduino na porcie szeregowym zaczynając od znaków „TO_ARDUINO:”. Zaczynając tak naszą wiadomość będziemy pewni, że Arduino zareaguje na nią.

Warto dodać, że tutaj również wszystkie możliwe błędy płyną na port szeregowy i w razie problemów debugging jest bardzo prosty.

4. Zdjęcia projektu

Powiadomienia dla użytkowników:



MikrokontroleryNa5 BOT Dziś o 00:10

Połączono z siecią WiFi!

Adres IP: 192.168.100.249



MikrokontroleryNa5 BOT Dziś o 00:23

Podłożono pod kotlem

Ilosc drewna: 3 Ilosc węgla: 2

Następne podłożenie przewidywane jest za: 180 minut

Czyli o godzinie: 03:23



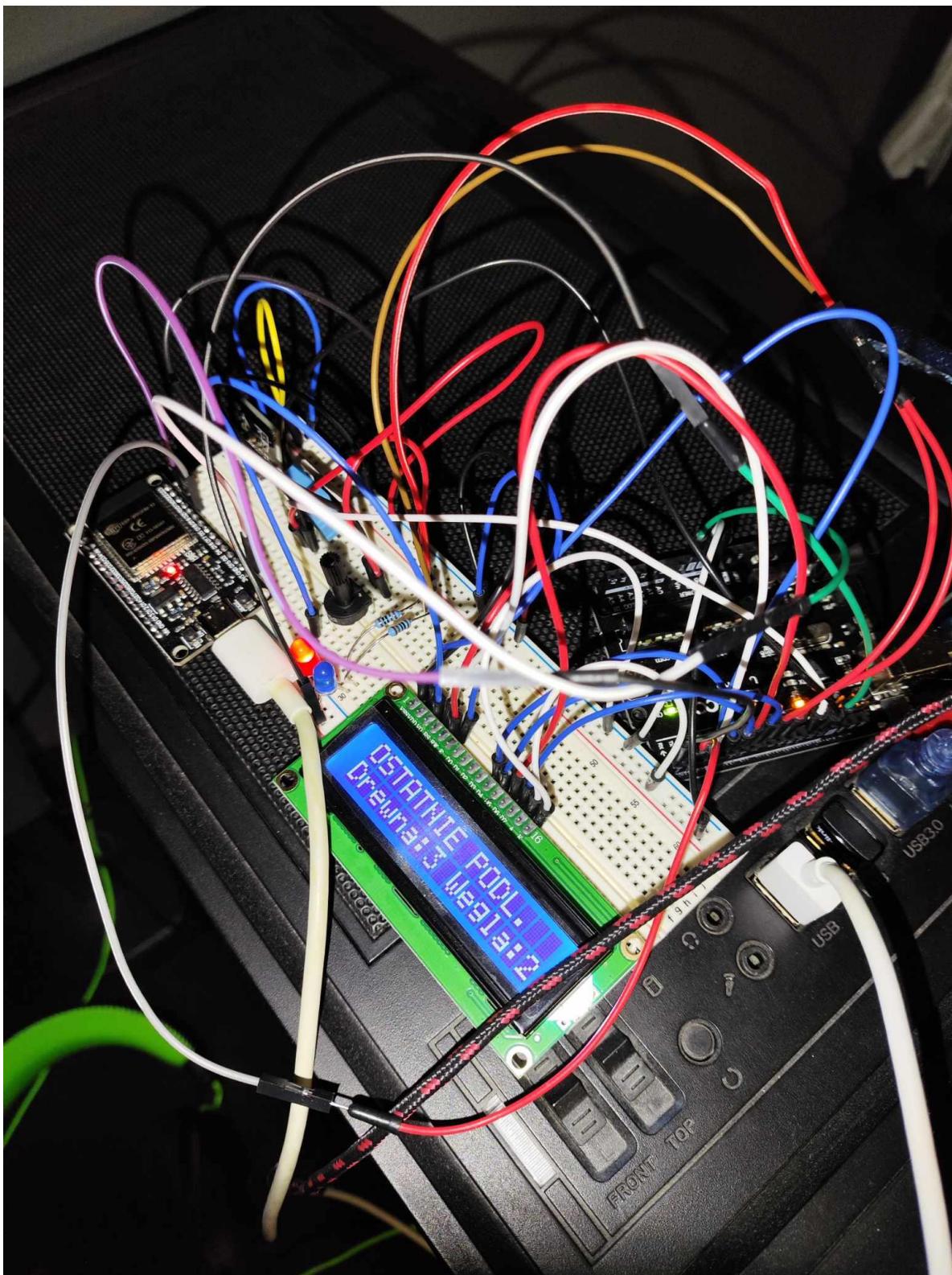
MikrokontroleryNa5 BOT Dziś o 00:34

Temperatura wzrosła z 22 na 23

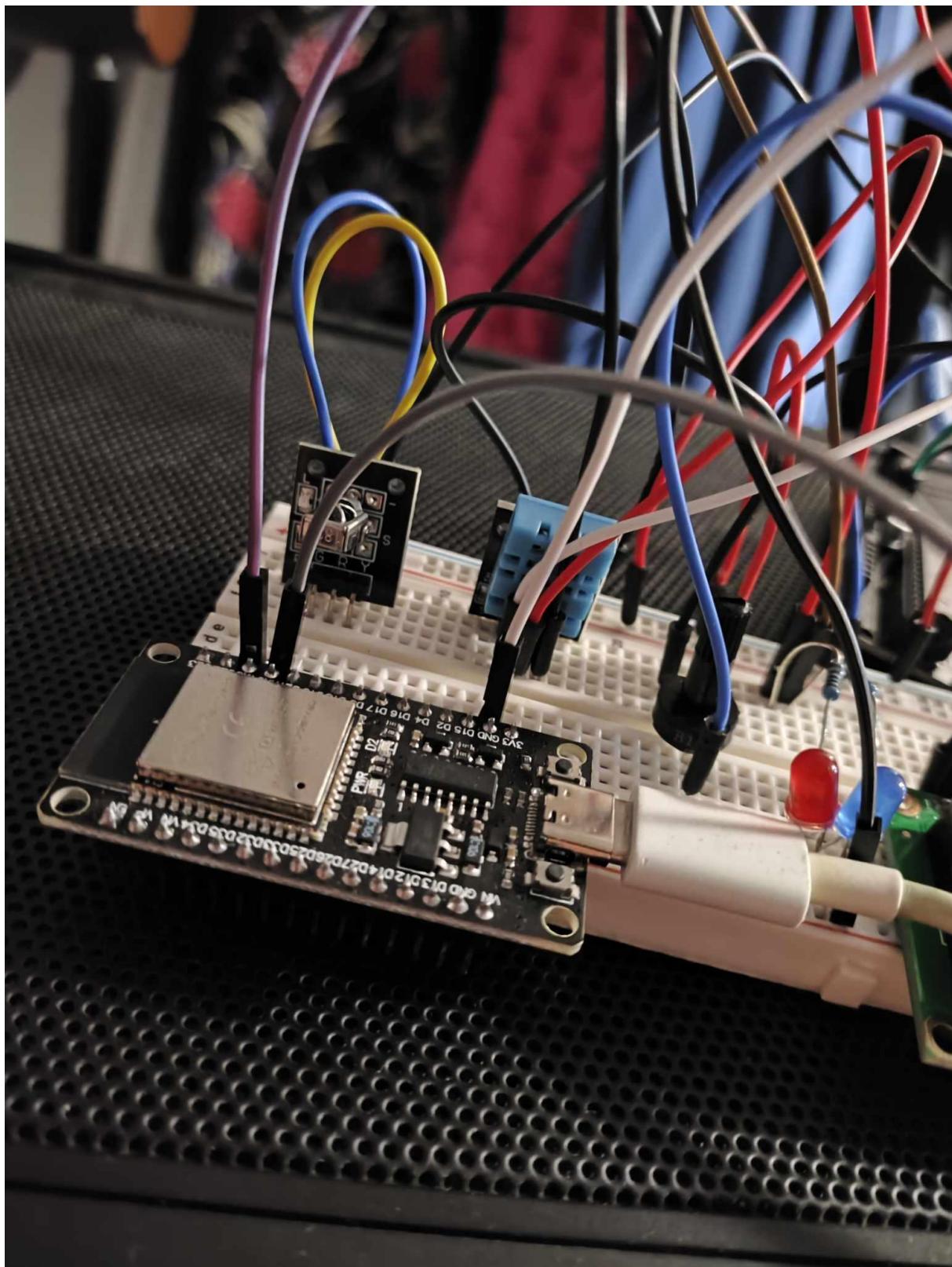
Temperatura wzrosła z 23 na 24

Temperatura zmalała z 24 na 23

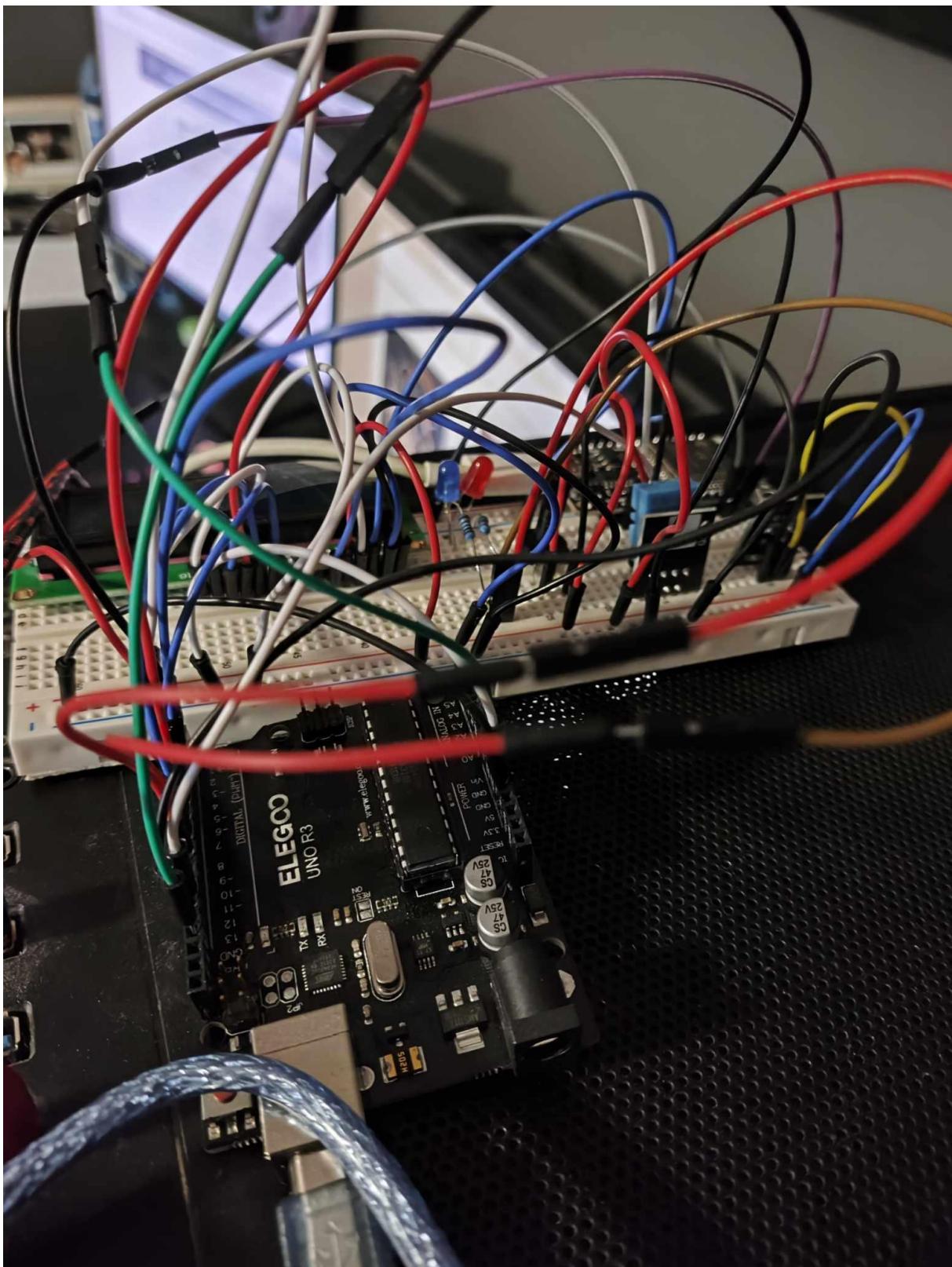
Projekt:



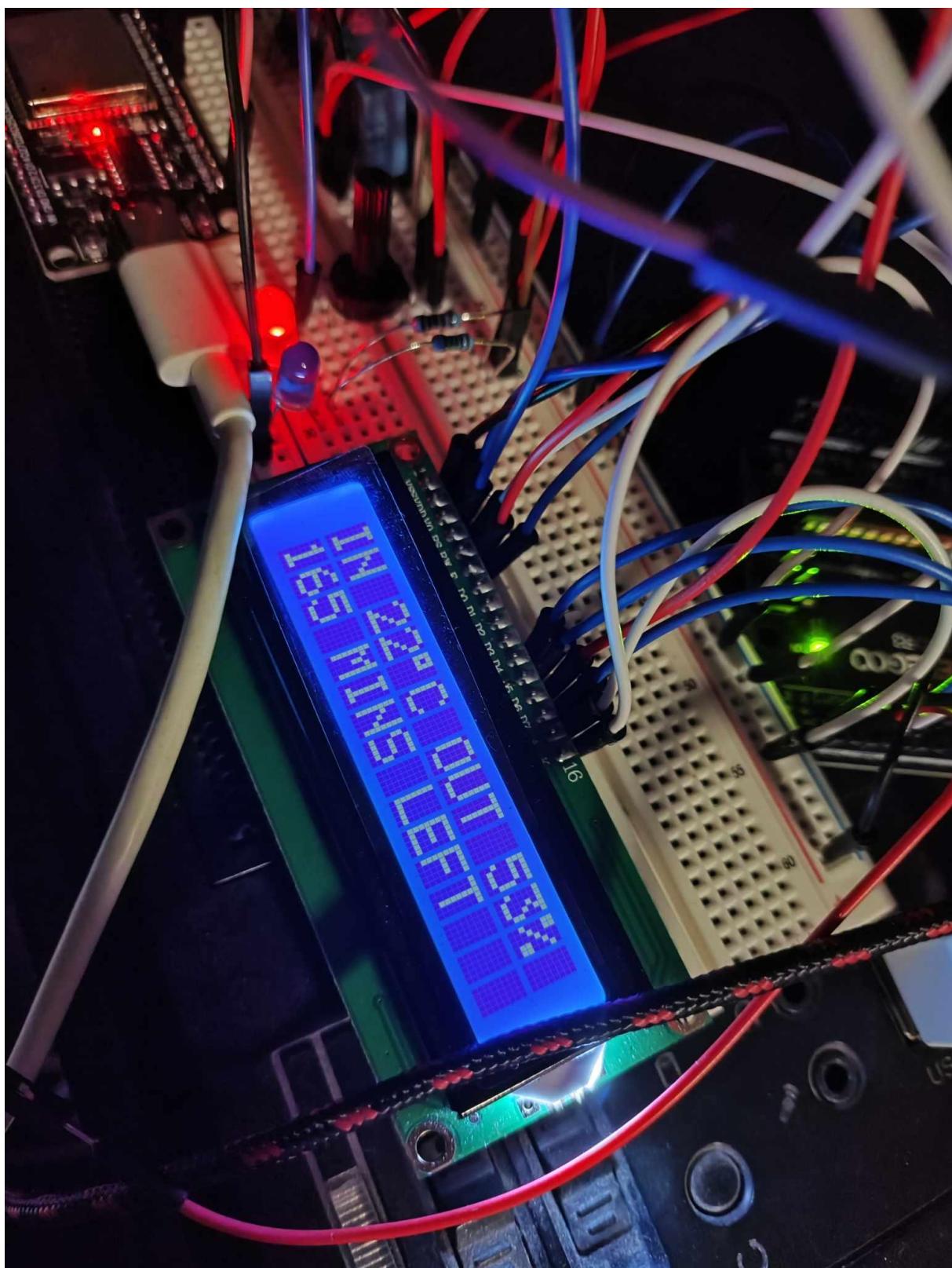
Czujnik wilgotności oraz podczerwieni + ESP:



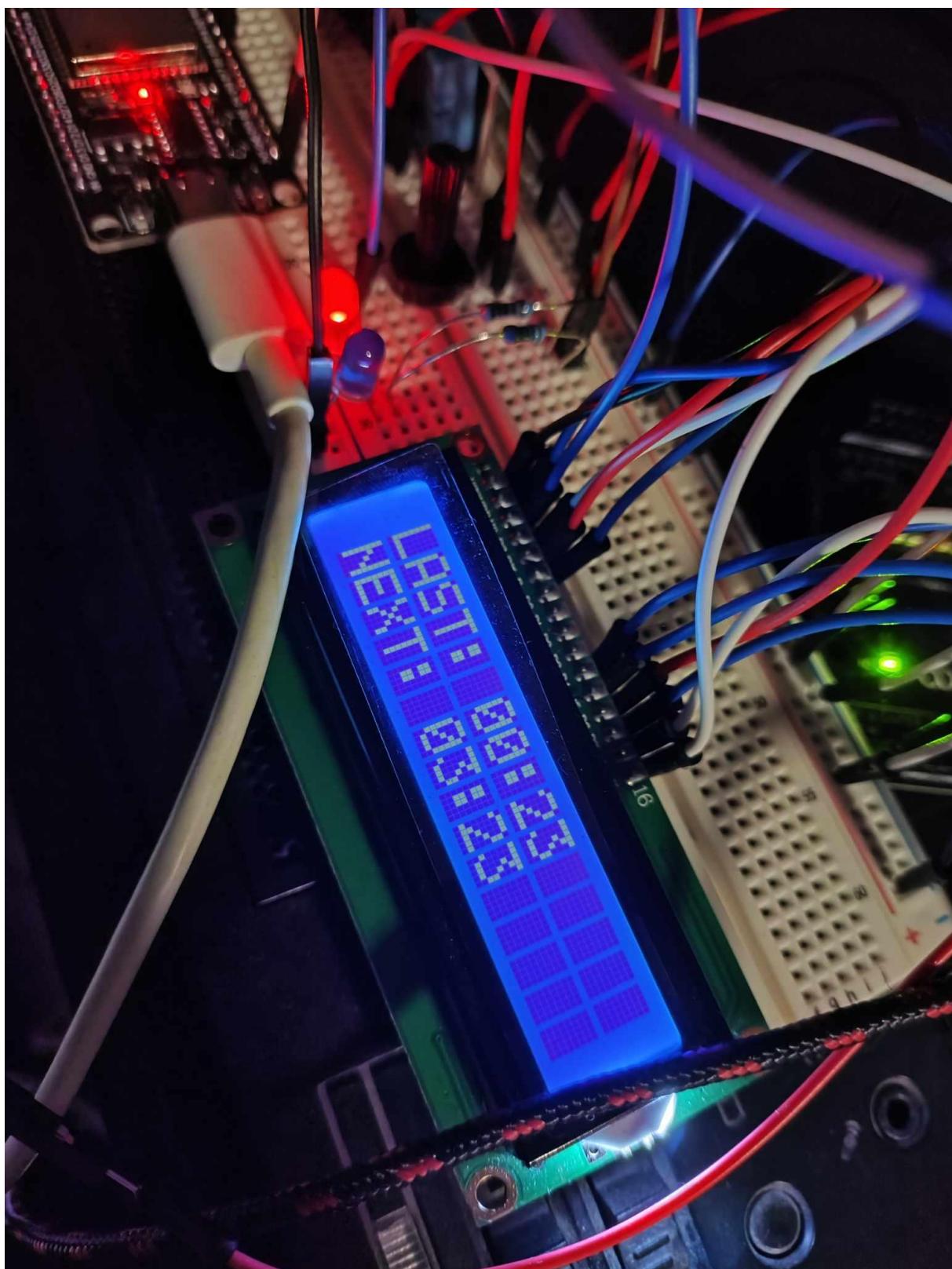
Widok od strony Arduino:



Monitor 1:



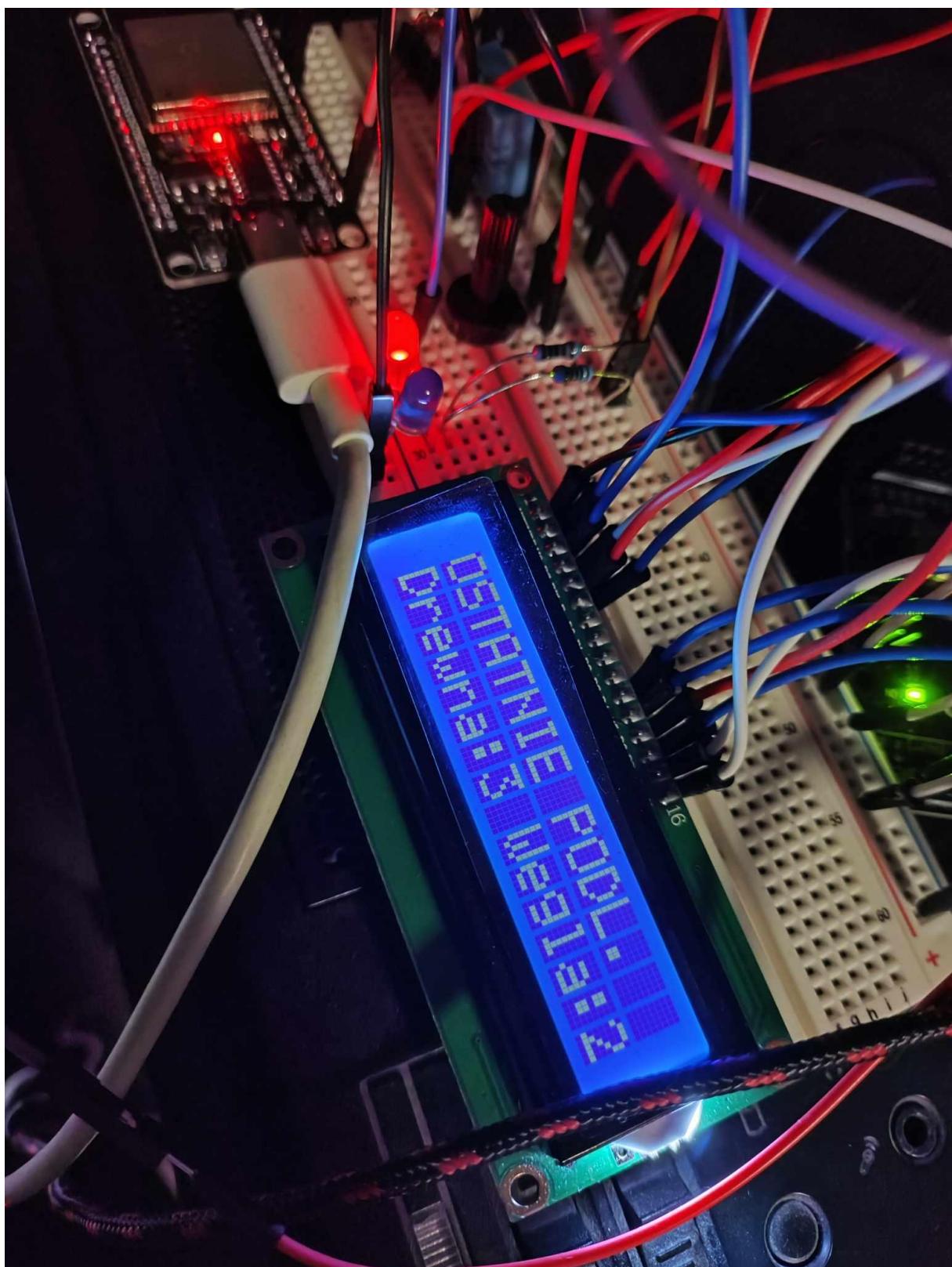
Monitor 2:



Monitor 3:



Monitor 4:



Podkładanie – ustalenie ilości drewna



Podkładanie – ustalanie ilości drewna – zmiana ilości



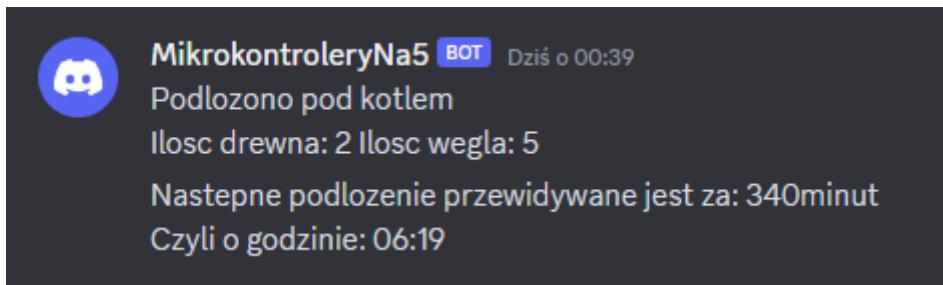
Podkładanie – ustalanie ilości węgla



Pomyślne wykonanie podłożenia



Powiadomienie po podłożeniu



Pilot używany do kontroli urządzenia



5. Podsumowanie

Projekt "Asystent Kontroli Ciepła" oferuje pomoc w zarządzaniu temperaturą, co przyczynia się do efektywniejszego wykorzystania energii oraz zwiększenia komfortu mieszkańców. Projekt ten stanowi punkt wyjścia do dalszych rozwiązań i integracji z innymi technologiami.