Internet Stvari

Projektna dokumentacija za Završni Ispit

1.Uvod

Sistem prototipa kontrole temperature u prostoriji i otvaranja vrata garaže se sastoji iz dva Arduino Uno R3 mikrokontrolera. Jedan Arduino kontroler kontroliše rad temperature tako što pokušava održavati temperaturu prostorije što bliže referentnoj koja je 23 stepena uz pomoć senzora temperature, ventilatora (DC motor) i grejača (LED dioda).

Ako je temperatura iznad 23 stepena pali se ventilator na onoliku snagu koliko je definsano sa 3 stanja:

- Razlika 0.0 stepeni iznad 23 stepena => 0% snage motora,
- Razlika između 0.0 – 3.0 stepena iznad 23 stepena => 50% snage motora,
- Razlika između 3.0 – 10.0 stepeni iznad 23 stepena => 70 % snage motora,
- Razlika iznad 10.0 stepeni iznad 23 stepena => 100 % snage motora

Kada je temperatura jednaka 23 stepena ili manja do 3 stepena, onda ne rade ni ventilator, ni grejač. A je temperatura niža za više od 3 stepena onda se simulira paljenje grejač paljenjem LED diode.

Temperatura se meri svakog minuta i podaci o temperaturi, snazi ventilatora i stanju grejača se šalju na ThingSpeak platformu u vidu 3 grafikona preko Wi-fi ESP8266 modula povezanog na arduino.

Na drugom arduinu je simulacija otvaranja garaže kada je objekat na udaljenosti između 5 i 15 cm. Za merenje razdaljine se koristi ultrazvučni senzor, dok se za simulaciju otvaranja garaže koristi Mikro servo motor pomeranjem na ugao od 90 stepeni. Kada objekat izađe iz definisanog okvira distance, servo motor se vrati u početni položaj na 0 stepeni.

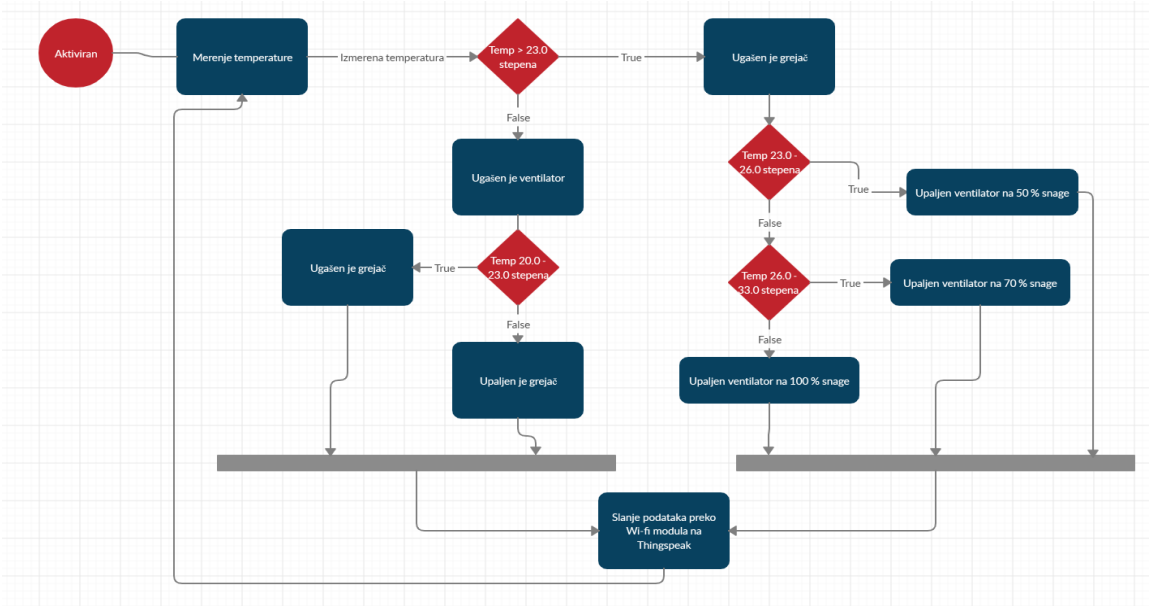
Obaveštenje o otvaranju garaže se šalje preko Wi-fi modula ESP8266 na privatni API od IFTTT Webhooks servisa na definisani email kao i u vidu notifikacije na mobilnom telefonu.

2. Dijagram stanja

Prvo je prikazan dijagram stanja prvog arduina koji kontroliše temperaturu u prostoriji. Imamo 3 glavna stanja:

- Temperatura 20.0 – 23.0 stepeni (Ne rade ni ventilator ni grejač),
- Temperatura > 23.0 stepena (Radi ventilator, ne radi grejač) i deli se na 3 podstanja:
    - Temperatura 23.0 – 26.0 stepeni (Radi ventilator na 50 % maksimalne snage),
    - Temperatura 26.0 – 33.0 stepeni (Radi ventilator na 70 % maksimalne snage),
    - Temperatura > 33.0 stepena (Radi ventilator na 100 % maksimalne snage)
- Temperatura < 20.0 stepena (Ne radi ventilator, radi grejač.

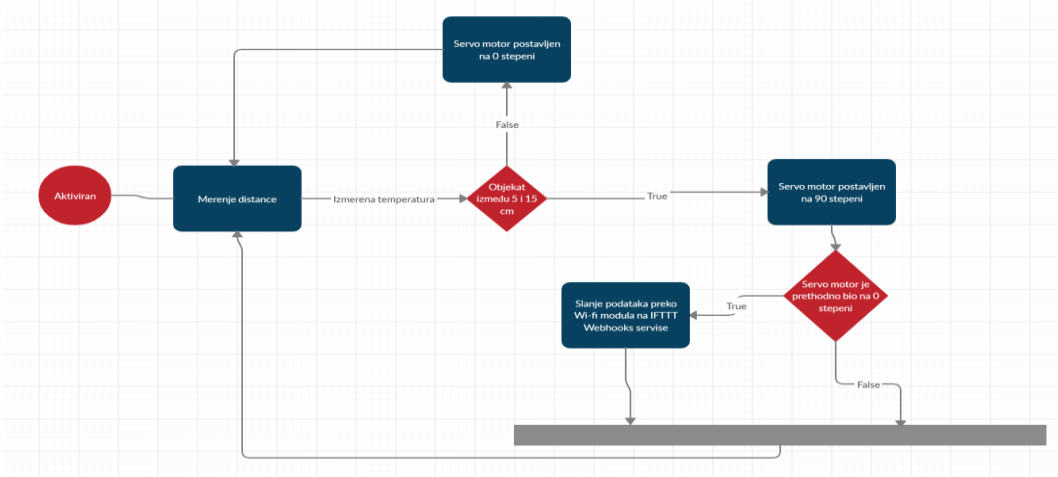
Ukupno ima 5 stanja kada se 2. Stanje podeli na 3 podstanja, kao što je i prikazanu na dijagramu ispod.



Na drugom Arduino se program deli na 2 glavna stanja:

- Objekat između 5cm i 15cm od senzora (Servo motor na 90 stepeni),
- Objekat < 5cm ili > 15cm distance od senzora (Servo motor na 0 stepeni)

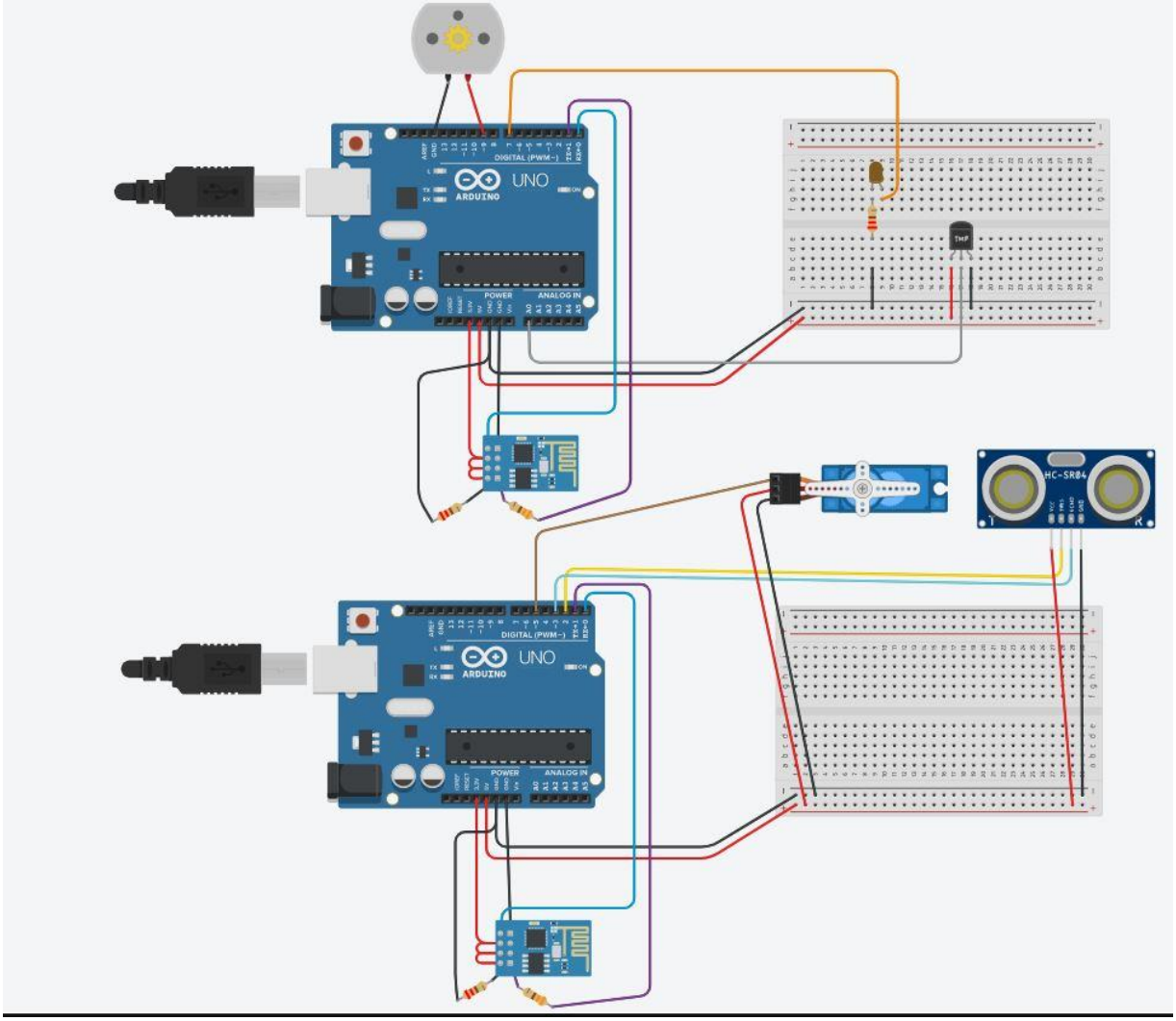
Dijagram stanja je prikazan ispod.



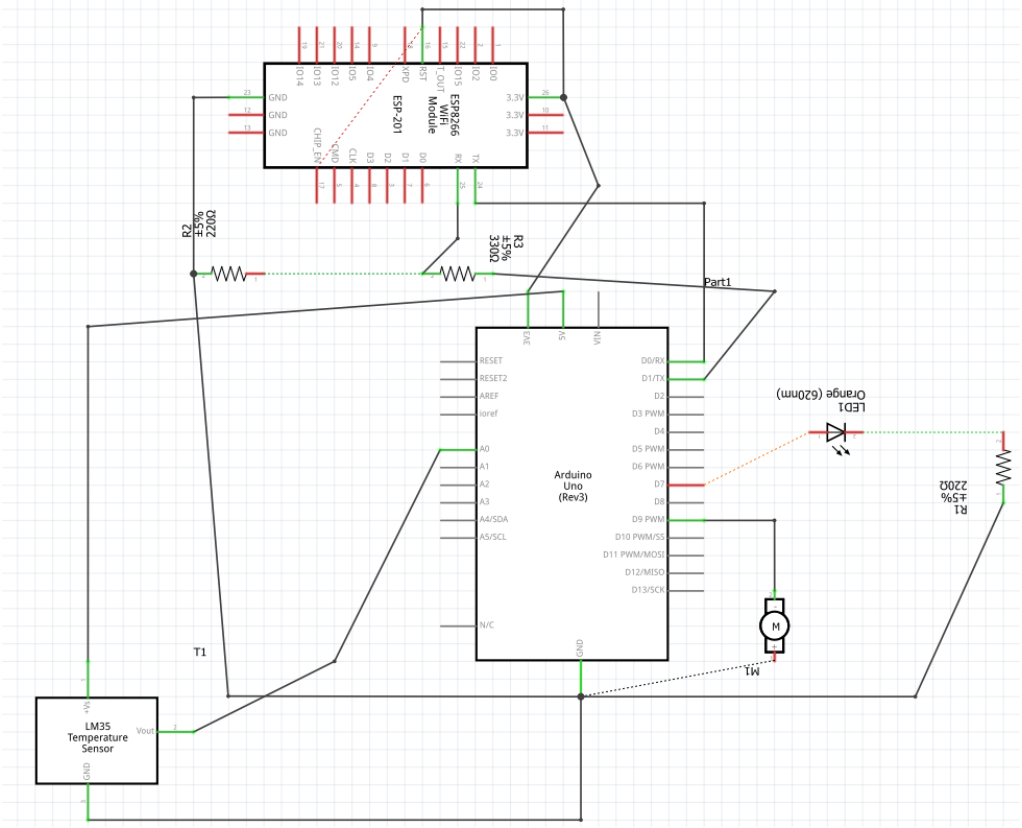
3.Fizička realizacija i šema električnog kola

Na slici ispod je prikaza fizička realizacija 2 Arduino mikrokontrolera koji su povezani sa prototipskom pločom koja je posrednik u električnom kolu sa senzorima i aktuatorima.
Za prvi Arduino se koristi Wi-fi modul ESP8266, DC motor, LED diode i LM35 analogni senzor temperature, kao i 3 otpornika (2 od 220 oma i 1 od 330 oma koji povezuje TX sa arduina na RX od Wi-fi modula. DC motor se može okretati u suprotnom smeru ako se obrnuto povežu GND i POWER pinovi, s obzirom da se unutar DC motora nalazi rotor koji posredstvom magnetnog polja rotira oko statira sa + i – polovima.

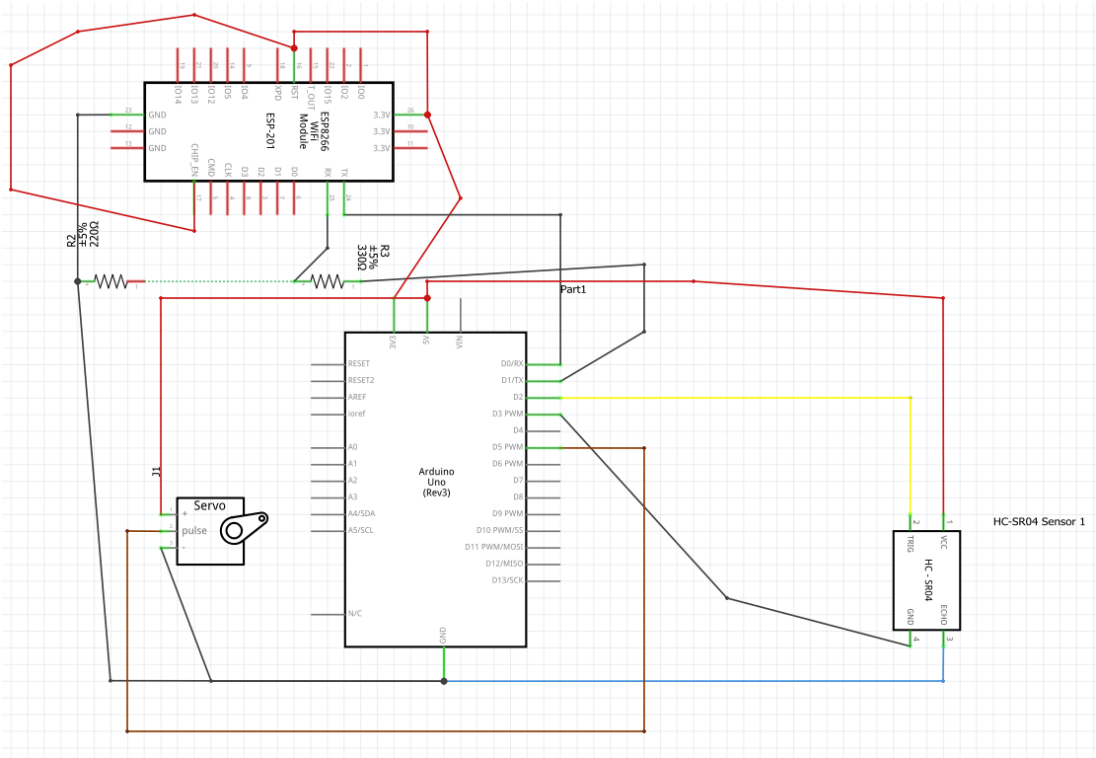
Na drugom arduinu je Wi-fi modul identično povezan kao na prvom i ovaj Wi-fi modul se napaja sa 3.3V napona s abilo kog Arduino mikrokontrolera. Takođe posredstvom prototipske ploče smo povezali Mikro Servo motor koji se može rotirati od 0 do 180 stepeni. Ultrazvučni sensor je takođe povezan preko prototipske ploče kao i TRIGGER i ECHO pinovi na digitalne pinove Arduina.



Ispod je prikaz šeme za Arduino koji kontroliše temperature u prostoriji uz pomoć senzora temperature LM35, LED diode (grejača) i DC motora (ventilatora).



Ispod je prikaz šeme električnog kola drugog arduina koji kontroliše rad garažnih vrata uz pomoć ultrazvučnog senzora i Mikro servo motora.



4. Analiza koda program, grafikona i obaveštenja

Ispod je prikaz delova koda prvog pa zatim drugog arduina od vrha ka dnu fajla.

1.Arduino

Na prvom arduinu smo definisali konstante za pinove koji su povezani sa analognim senzorom temperature, LED diodom i DC motorom. Zadan je period merenja od 60000 milisekundi što je 1 minut, referentna temperature od 23 stepena, maksimalna snaga DC motora koja je 255.
Takođe su zadati parametri za konekciju na Wi-fi preko ESP8266 modula, link ka ThingSpeak API sa API ključem i HTTP portom.

```
#define HEAT 7
#define TEMP_SENSOR A0
#define COOLER 9
int measurePeriod = 60000; // 1 minute.
const double refTemp = 23.0;
const int maxCoolerPower = 255;
float temp, coolerPow;
int heatState;
unsigned long prevMillis;
String ssid = "Simulator Wifi";
String password = "";
String link = "https://api.thingspeak.com";
const int port = 80;
String uri = "/update?api_key=TAQQIM5C7XWY8LB8";
```

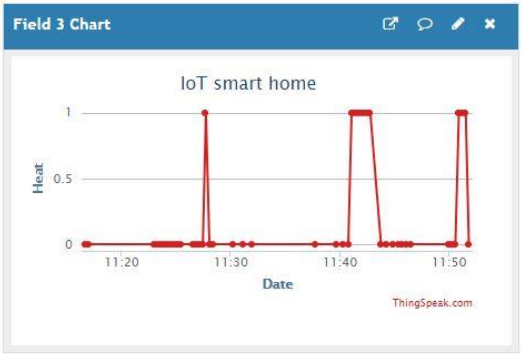
U **setup()** funkciji se vrši povezivanje na Wi-fi uz ssid i lozinku, kao uspostavljanje veze preko TCP. Na kraju su setovani modovi za pinove. Senzor temperature je postavljeni kao **INPUT**, dok su za LED diodu (grejač) i DC motor postavljeni kao **OUTPUT.**

```
void setup(){
    Serial.begin(115200);
    Serial.println("AT");
    if (!Serial.find("OK")){
      Serial.println("---------Error TEST---------");
    };

    Serial.println("AT+CWJAP=\"" + ssid + "\",\"" + password + "\"");
    if (!Serial.find("OK")){
      Serial.println("---------Error CWJAP---------");
    };
    Serial.println("AT+CIPSTART=\"TCP\",\"" + link + "\"," + port);
    if (!Serial.find("OK")){
      Serial.println("---------Error CIPSTART---------");
    };
    pinMode(HEAT, OUTPUT);
    pinMode(TEMP_SENSOR, INPUT);
    pinMode(COOLER, OUTPUT);
}
```

Na kraju u glavnoj loop() funkciji merimo temperature na svakih minut. Temperaturu čitamo sa analognog pina koju pretvaramo u milivolte množeći sa 5V i onda deleći sa 1024 i od tog rezultata i prebacujemo sa milivolta u temperature u celzijusima.
Zatim proveravamo da li je temperature veća ili nije. Ako jeste onda stavljamo da grejač bude ugašen i proveravamo opet kolika je razlika između referentne i izmerene temperature kako bi znali na koju snagu da upalimo ventilator.
Ako je manja izmerena temperature od referentne onda proveravamo kolika je ta razlika i ako je veća od 3 stepena onda se pali grejač.
Na kraju podatke o temperature, snazi ventilatora i stanju grejača pakujemo u HTTP paket koji se šalje serijskom vezom do Wi-fi modula.

```
void loop(){
    if(millis() - prevMillis >= measurePeriod){
        prevMillis = millis();

        // Calculate temperature in C by converting Analog signal.
        temp = ((analogRead(TEMP_SENSOR) * 0.00488) - 0.5) * 100;
        if(temp > refTemp){
            heatState = LOW;
            digitalWrite(HEAT, heatState);
            if(temp < refTemp + 3){
                coolerPow = 50;
            } else if (temp < refTemp + 10){
                coolerPow = 70;
            } else {
                coolerPow = 100;
            }
            analogWrite(COOLER, maxCoolerPower*(coolerPow/100.0));

        } else { // temp <= 23C
            coolerPow = 0;
            analogWrite(COOLER, coolerPow);
            if (temp < refTemp - 3){
                heatState = HIGH;
            } else { // temp == 23C
                heatState = LOW;
            }
            digitalWrite(HEAT, heatState);
        }

        // Create HTTP request message.
        String httpRequest = "GET " + uri +
          "&field1="+temp+"&field2="+coolerPow+"&field3="+heatState
          + " HTTP/1.1\r\nHost: " + link + "\r\n\r\n";
        int length = httpRequest.length();

        // Info for Wifi module about packet length.
        Serial.print("AT+CIPSEND=");
        Serial.println(length);

        // Send HTTP GET request to ESP8266.
        Serial.print(httpRequest);
        if (!Serial.find("SEND OK\r\n")){
        }
    }
}
```

Ispod je prikaz podataka na ThingSpeak platformi sa 3 grafikona. Prvi prikazuje kretanje temperature kroz vreme, drugi snagu ventilatora i treći stanja grejača.



2.Arduino

Na drugom Arduinu se vrši kontrola garaže korišćenjem mikro servo motor ai ultrazvučnog senzora koji su povezani na pinove čiji su brojevi definisani kao konstante. Takođe neophodno je uvesti biblioteku za Servo motor **<Servo.h>** i krerati prazan objekat **Servo.** Promenljivu **garageOpened** koristimo kao fleg da se obaveštenje o otvaranju šalje samo nakon što je objekat prvi put ušao u predviđen proctor (između 5 i 15cm). Koristimo isti ssid, bez lozinke, ali je uri za priustupanje okidaču od IFTTT Webhooks servisa sada definisan kao i link za IFTTT.

```
#include <Servo.h>
#define SERVO 5
#define TRIGGER 2
#define ECHO 3
long duration; // Variable for the duration of sound wave travel.
float distance;
bool garageOpened = false; // Variable to send IFTTT request only once
String ssid = "Simulator Wifi";
String password = "";
String link  = "https://maker.ifttt.com";
const int port = 80;
String uri = "/trigger/garage_opened/with/key/dZj6Bwm8AeLdMt3SC7TevIq9it3-oevKOurZNNMgQhx";
Servo servo;
```

U **setup()** funkciji se vrši povezivanje na Wi-fi uz ssid i lozinku, kao uspostavljanje veze preko TCP. Na kraju su setovani modovi za pinove. Ultrazvučni senzor je postavljeni sa **ECHO** kao **INPUT** jer odatle dobijamo nazad podatak o trajanju dok nismo dobili nazad zvučni signal, dok je TRIGGER postavljen kao **OUTPUT** jer Arduino šalje izlazni signal ka senzoru koji će okinuti da sensor pošalje zvučni signal ka objektu**.** Takođe smo na kraju pomoću funkcije **attach()** nakačili servo motor.

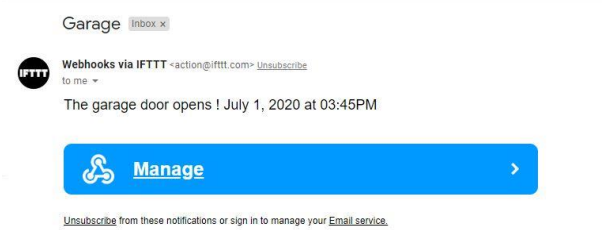
```
void setup(){
  Serial.begin(115200);
  Serial.println("AT");
  if (!Serial.find("OK")){
    Serial.println("---------Error TEST---------");
  };
  Serial.println("AT+CWJAP=\"" + ssid + "\",\"" + password + "\"");
  if (!Serial.find("OK")){
    Serial.println("---------Error CWJAP---------");
  };
  Serial.println("AT+CIPSTART=\"TCP\",\"" + link + "\"," + port);
  if (!Serial.find("OK")){
    Serial.println("---------Error CIPSTART---------");
  };
  pinMode(TRIGGER, OUTPUT);
  pinMode(ECHO, INPUT);
  servo.attach(SERVO);
}
```

U glavnoj loop() funkciji prvo šaljemo nizak napon na TRIGGER pin pa nakon kratke pause šaljemo visok napon, pa opet nizak nakon 10 milisekundi da bi smo inicirali pokretanje zvučnog signala iz senzora. Nakon što se zvuk odbije i vrati dobijamo sa ECHO vremenski period koji je protekao nakon što je TRIGGER inicirao zvučni signal. Prema formuli da zvuk putuje 340m/s i kad prebacimo u centimetr i podelimo sa 2 (jer je zvuk 2 puta prešao put od objekta i odbio se nazad) pa dobijemo koliko je objekat udaljen u centimetrima. Zarim proveravamo da li je objekat između 5 i 15cm i ako jeste postavljamo servo motor na 90 stepeni što simulira otvaranje garažnih vrata. Takođe nakon toga proveravamo da li su pre toga vrata garaže bila otvorena, jer ako nisu onda se šalje jednom HTTP paket preko Wi-fi modula pa prema definisanim parametrima ka IFTTT servisu koji će poslati mejl i notifikaciju

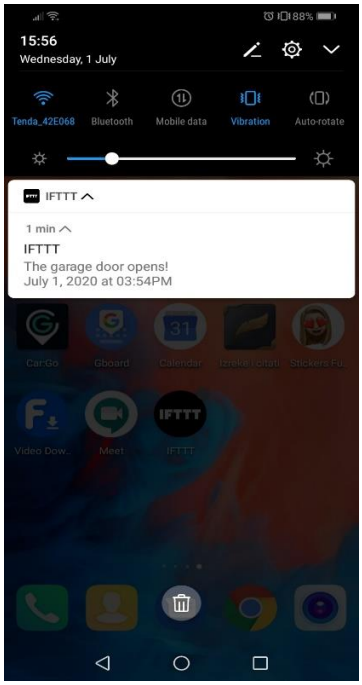
na mobilnom telefonu. I konačno, ako objekat nije na udaljenosti kada bi trebala da su otvorena vrata onda se upisuje 0 kao vrednost za stepene ugla servo motor ai resetuje se promenljiva **garageOpened**.

```
void loop(){
    digitalWrite(TRIGGER, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIGGER, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIGGER, LOW);
    duration = pulseIn(ECHO, HIGH);

    // Calculating the distance.
    // Speed of sound wave divided by 2 (go and back).
    distance = duration * 0.034 / 2;
    if(distance >= 5.0 && distance <= 15.0){
        servo.write(90);
        if(!garageOpened){
            garageOpened = true;

            // Create HTTP request message.
            String httpRequest = "GET " + uri +
              " HTTP/1.1\r\nHost: " + link + "\r\n\r\n";
            int length = httpRequest.length();

            // Info for Wifi module about packet length.
            Serial.print("AT+CIPSEND=");
            Serial.println(length);

            // Send HTTP GET request to ESP8266.
            Serial.print(httpRequest);
            if (!Serial.find("SEND OK\r\n")){
            }
        }
    } else {
        garageOpened = false;
        servo.write(0);
    }
}
```

Kada IFTTT Webhook servis pošalje mejl da su vrata otvorena, to izgleda kao na slici ispod.



Dok drugi IFTTT Webhook šalje notifikaciju na telefonu.



3. Python program na PC

Na PC računaru se nalazi Python program koji će se ulogovati na definisani mejl preko IMAP protokola i pokušati da nađe mejl sa naslovom IZVESTAJ koji nije pregledan, a zatim učitati temperature poslednjih 7 merenja sa ThingSpeak i poslednjih 7 sati sa Weather2Umbrella odakle su izvučene temperature uz pomoć Web Scraping tehnike i BeautifulSoup biblioteke i kreiraiti grafik sa ta 2 uzorka koji će sačuvati kao temperature.png sliku. Zatim će učitati i obraditi podatke sa ThingSpeak o prosečnoj snazi ventilatora u procentima od poslednjeg izveštaja, broj paljenja grejača i otvaranja garaže prema broju neviđenih mejlova od IFTTT. Sve podatke će spakovati u Body od mejla koji će se poslati onom ko je poslao mejl sa nalsovom IZVESTAJ.

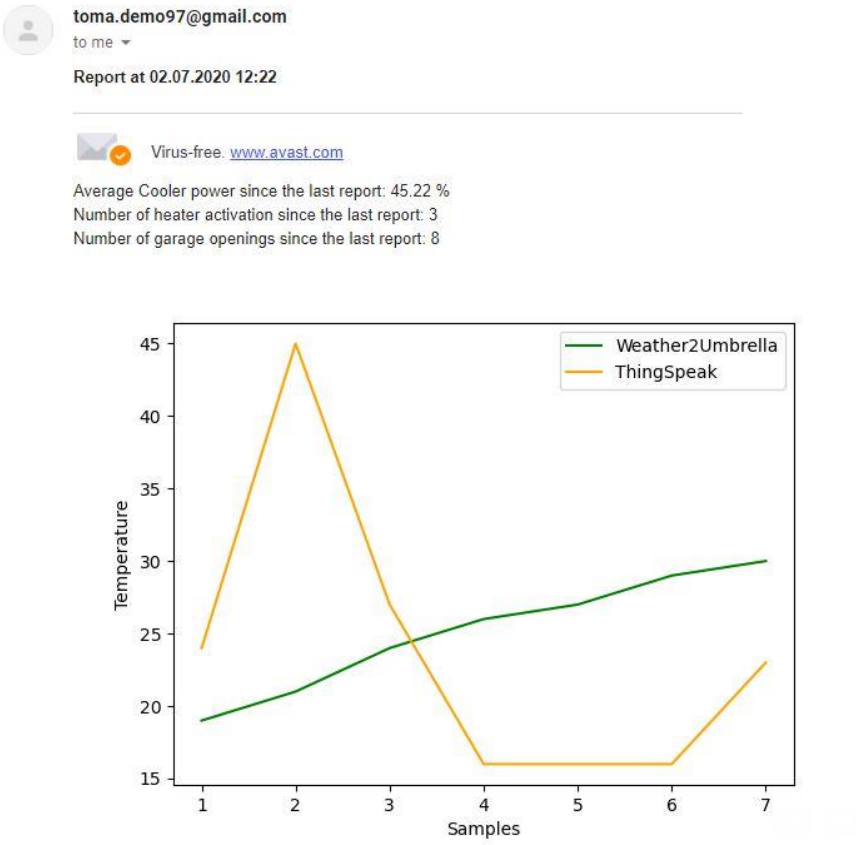
```python
import smtplib
import imaplib
import email as emailDecoder
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart
import matplotlib.pyplot as plt
import requests
from bs4 import BeautifulSoup
import time

def getEmailSender(brojMejla):
    _, msg = imap.fetch(str(int(brojMejla)), "(RFC822)")
    for response in msg:
        if isinstance(response, tuple):
            msg = emailDecoder.message_from_bytes(response[1])
            from_who = msg.get("From").split()[-1].replace("<","").replace(">","")
            return from_who

def fetchtempWeather2Umbrella():
    respWeather2Umbrella = requests.get("https://www.weather2umbrella.com/vremenska-prognoza-beograd-srbija-sr/trenutno")
    weather_per_hours = BeautifulSoup(respWeather2Umbrella.text, "html.parser").find('div', attrs={'class':'weather_per_hours'})
    hourly_temp = weather_per_hours.find_all('div', attrs={'class':'col-xs-2 col-sm-1 hourly_temp'})
    tempList = []
    for temp in hourly_temp:
        tempList.append(int(temp.p.text.replace("°","")))
    hours = int(time.strftime("%H"))
    start = (hours - 6) if (hours - 6) > 0 else 1
    tempWeather2Umbrella = []
    for i in range(start - 1, hours):
        tempWeather2Umbrella.append(tempList[i])

    # Add here to look for last 7 hours with previous day.
    if len(tempWeather2Umbrella) == 0:
        tempWeather2Umbrella = [0] * 7
    return tempWeather2Umbrella

def fetchTempThingSpeak():
    respThingSpeak= requests.get("https://api.thingspeak.com/channels/1092688/feeds.json?api_key=ZR7N919RTEZO6MT7&results=7")
    values = respThingSpeak.json()['feeds']
    tempThingSpeak = []
    for val in values:
        tempThingSpeak.append(int(float(val['field1'])))
    return tempThingSpeak
```

```python
def plotTemperatures(tempWeather2Umbrella, tempThingSpeak):
    f = plt.figure()
    plt.plot(range(1,8), tempWeather2Umbrella, color="green", label="Weather2Umbrella")
    plt.plot(range(1,8), tempThingSpeak, color="orange", label="ThingSpeak")
    plt.ylabel("Temperature")
    plt.xlabel("Samples")
    plt.legend()
    f.savefig("temperature.png")

def getCoolerHeaterAvg(lastReportNum):
    respThingSpeak= requests.get("https://api.thingspeak.com/channels/1092688/feeds.json?api_key=ZR7N919RTEZO6MT7")
    values = respThingSpeak.json()['feeds']
    coolerAllValues = []
    heaterAllValues = []
    for val in values:
        coolerAllValues.append(int(float(val['field2'])))
        heaterAllValues.append(int(val['field3']))
    afterLastReportCoolerValues = []
    afterLastReportHeaterValues = []
    for indx in range(lastReportNum, len(coolerAllValues)):
        afterLastReportCoolerValues.append(coolerAllValues[indx])
        afterLastReportHeaterValues.append(heaterAllValues[indx])
    lastReportNum = len(heaterAllValues)
    countHeatOn = 0
    for i in range(0, len(afterLastReportHeaterValues)):
        if afterLastReportHeaterValues[i] == 1 and (i == 0 or afterLastReportHeaterValues[i-1] == 0):
            countHeatOn += 1
    avgCoolerPower = round(sum(afterLastReportCoolerValues)/len(afterLastReportCoolerValues), 2) if len(afterLastReportCoolerValues) > 0 else 0
    return avgCoolerPower, countHeatOn, lastReportNum

def getNewGarageOpeningsCount(imap):
    response, newGarageOpenings = imap.search(None,'SUBJECT "Garage" UNSEEN')
    print("IMAP response searching for mail subject 'Garage': " + response)
    countNewGarageOpenings = len(newGarageOpenings[0].split())
    for i in newGarageOpenings[0].split():
        imap.store(i, '+FLAGS', '\\SEEN')
    return countNewGarageOpenings
```

```python
def createReport(email, password, to, avgCoolerPower, countHeatOn, countNewGarageOpenings):
    report = MIMEMultipart()
    report["Subject"] = "RESPORT RESPONSE"
    report["From"] = email
    report["To"] = to
    mimeTextsValues = {time.strftime("%d.%m.%Y %H:%M"): "<b>Report at {}</b><br>",
                       avgCoolerPower: "Average Cooler power since the last report: {0:.2f} %<br>",
                       countHeatOn: "Number of heater activation since the last report: {0:d}<br>",
                       countNewGarageOpenings: "Number of garage openings since the last report: {0:d}<br>" }
    for key in mimeTextsValues.keys():
        report.attach(MIMEText(mimeTextsValues[key].format(key), 'html'))
    report.attach(MIMEText("<img src='cid:image1'><br>",'html'))
    with open("temperature.png", 'rb') as filePlot:
        imgPlot = MIMEImage(filePlot.read())
        imgPlot.add_header('Content-ID', '<image1>')
    report.attach(imgPlot)
    smtpServer = smtplib.SMTP('smtp.gmail.com', 587)
    smtpServer.starttls()
    smtpServer.login(email, password)
    smtpServer.sendmail(email, to, report.as_string())
    smtpServer.quit()

def processDataAndCreateReport(request_mail_num, lastReportNum, imap, email, password):
    to = getEmailSender(request_mail_num)
    print("Fetching and processing temperatures from Thingspeak and Weather2Umbrella...")
    tempWeather2Umbrella = fetchtempWeather2Umbrella()
    tempThingSpeak = fetchTempThingSpeak()
    plotTemperatures(tempWeather2Umbrella, tempThingSpeak)
    print("Fetching and processing cooler and heater average values after last report...")
    avgCoolerPower, countHeatOn, lastReportNum = getCoolerHeaterAvg(lastReportNum)
    print("Getting garage openings emails since last report...")
    countNewGarageOpenings = getNewGarageOpeningsCount(imap)
    print("New report is creating...")
    createReport(email, password, to, avgCoolerPower, countHeatOn, countNewGarageOpenings)
    return lastReportNum
```

```python
if __name__ == "__main__":
    lastReportNum = 0
    email = "toma.demo97@gmail.com"
    password = "Tomademo97+"
    while True:
        imap = imaplib.IMAP4_SSL('imap.gmail.com')
        imap.login(email, password)
        print("Logged in!")
        imap.select("INBOX")
        response, report = imap.search(None, 'SUBJECT "IZVESTAJ" UNSEEN')
        print("IMAP response searching for mail subject 'IZVESTAJ': " + response)
        for request_mail_num in report[0].split():
            lastReportNum = processDataAndCreateReport(request_mail_num, lastReportNum, imap, email, password)
```

Prikaz poslatog izveštaja na mejl je prikazan ispod.



5. Zaključak i moguća nadogradnja

Sistem predstavlja samo prototip za simulaciju kontrole temperature pomoću senzora temperature, simuliranog grejača pomoću LED diode i simuliranog ventilatora pomoću DC motora, s druge strane simulacija otvaranje garaže uz približavanje objekta i dobijanje obaveštenja i izveštaja mejlom. Stoga se ovakav sistem može primeniti u praktične svrhe uz adaptaciju softvera i zamenu za adektvatan hardver.

Ovaj sistem se može proširiti sa dodatnim senzorima i aktuatorima kako bi zajedno formirali jedan pametan ugrađen sistem za kontrolu potrošnju struje (klime, grejanja, mehanizacije) i vode. S druge strane bi servis koji obrađuje podatke prikupljene iz Cloud platformi mogao preko algoritama Mašinskog učenja na osnovu određenih šablona da tačno unapred predvidi potrošnju struje i kretanje određenih vrednosti sa što minimalnijom greškom.

Takođe je potrebno naglasiti da su ovakve sisteme nepohodni najnapredniji kriptološki algoritmi i protokoli s obzirom da se sve kontroliše preko mreže da je prilikom napada moguće dovesti sistem ili okolinu u stanje havarije, pa čak i ugroziti ljudske živote.