Mjerenje parametara okoliša pomoću Arduino senzora

Čečura, Stjepan

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Stjepan Čečura

MJERENJE PARAMETARA OKOLIŠA POMOĆU ARDUINO SENZORA

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Stjepan Čečura

Matični broj: 0016142880

Studij: Informacijski sustavi

Mjerenje parametara okoliša pomoću Arduino senzora

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Ivan Magdalenić

Varaždin, rujan 2023.

Stjepan Čečura

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je mjerenje određenih parametara okoliša, poglavice temperature i vlage te drugih i prikazati mjerenja na korisniku jednostavan i pristupačan način. U radu su obrađeni korišteni alati i tehnologije te potom i sustav zajedno s njegovim dijelovima slanja mjerenja pomoću Arduino mikro kontrolera i interneta te prikazom podataka na web pretraživačima pomoću Google Firebase usluge.

Ključne riječi: Arduino; Internet stvari(IoT); Firebase; Web aplikacija; mikro kontroler; senzori okoliša

Sadržaj

1. Uvod

U završnom radu je obrađena izrada web aplikacije preko koje je moguće pratiti temperaturu, vlagu, pritisak, svjetlost i buku okoliša u koji su senzori postavljeni zajedno s pratećim mikro kontrolerom. Senzori mjere navedene parametre i pomoću mikro kontrolera ih WiFi vezom šalju internetom na Google Firebase bazu podataka. Web aplikacija podatke preuzima iz baze podataka i prikazuje ih u obliku individualnih grafova za svaki parametar pored kojih se nalazi ikona sa zelenom ili crvenom pozadinom u ovisnosti o tome je li najnovije mjerenje iznad zadane vrijednosti za upozoravanje te poljem za unos nove vrijednosti ispod ikone.

S povećanom pristupačnošću uređajima poput mikro kontrolera i tehnologijama web hostinga osobama koje imaju manje iskustva ili stručnosti s njima, rad pokazuje kako je moguće napraviti sustav za osobnu upotrebu koristeći lako dostupne resurse na internetu. Prikazan je sustav koji mjeri određene vrijednosti i samo ih prikazuje, no raznim promjenama i nadogradnjama je moguće prilagoditi sustav da mjeri druge vrijednosti ili vrši različite funkcije poput zalijevanja biljki na osnovu vlažnosti tla, otvaranja i zatvaranja vrata ili prozora na prema nekom uvjetu ili interaktivnu igru.

2. Metode i tehnike rada

U ovom poglavlju su navedeni korišteni alati i servisi, prvo za Arduino podsustav, potom za web aplikaciju te onda uređivači koda i tehnologije.

2.1. Arduino

Arduino je tvrtka koja je postala popularna zbog jednostavnih, jeftinih i lako upotrebljivih proizvoda, od mikro kontrolera i senzora do edukativnih paketa s raznim temama koje dizajniraju, proizvode i podržavaju.

2.1.1. Arduino UNO WiFi Rev.2

Arduino UNO WiFi Rev.2 je mikro kontroler tvrtke Arduino koji je većinski identičan najpopularnijem mikro kontroleru tvrtke, Arduino Uno Rev.3, s dodatnom funkcionalnosti WiFi i Bluetooth zbog kojih je i izabran za izradu ovog rada kako bi se mjerenja senzora mogla slati na bazu podataka ili prikazati na serveru koji je moguće pokretati sa samog mikro kontrolera.

2.1.2. Arduino Sensor Kit

Dodatno uz mikro kontroler su potrebni i senzori koji će zapravo vršiti mjerenje parametara okoliša i za koju ulogu je odabran Arduino Sensor Kit. Navedeni set je odabran zbog njegove jednostavnosti korištenja koja je posljedica jednostavnih primjera na popratnoj Internet stranici i principa spajanja senzora time što ploču na kojoj se senzori nalaze uklopimo na mikro kontroler ili spojimo kablovima ako su odvojeni od ploče. Na ploči se nalaze LED lampica, zvučnik, OLED ekran, gumb, potenciometar, senzor svjetlosti, senzor zvuka, senzor temperature, senzor pritiska zraka i akcelerometar.

2.2. Google Firebase

Web aplikacija preko koje se prikazuju mjerenja i upozorenja je napravljena preko Google Firebase koji je platforma za razvoj aplikacija s raznim uslugama poput Cloud Firestore, Cloud Functions, Authentication te Hosting i Realtime Database koji su jedine dvije korištene za pružanje usluga web aplikacije u ovom radu. Google Firebase je izabran za ovaj projekt poglavice zbog jednostavnosti kreiranja računa i započinjanja rada, minimalne potrebe za radom te planova plaćanja od kojih je osnovni plan besplatan i ne zahtijeva unošenje kreditne kartice, a plaćani plan naplaćuje samo onoliko koliko korisnik iskoristi.

2.2.1. Google Firebase Realtime Database

Realtime Database je baza podataka koja se nalazi na Cloudu i bazira se na NoSQL pristupu pohranjivanju podataka koji je bio prikladan za projekt koji koristi jednu listu objekata koji sadrže podatke o mjerenjima te jedan objekt u kojem se pohranjuju vrijednosti za provjeru visokih mjerenja. Umjesto tablica podataka, podatci se pohranjuju kao JSON koji je moguće izvesti za druge svrhe.

2.2.2.Google Firebase Hosting

Web aplikacija je sadržana na Firebase Hosting servisu zbog već navedenih razloga jednostavni i pristupačnosti, poglavice mogućnosti objavljivanja stranice jednom naredbom nakon odrađene pripreme koja je drugi razlog izbora ovog servisa jer ne zahtijeva kompleksno namještanje servera nego instaliranje Firebase CLI, Node.js i Webpack.

2.3. Uređivači koda

2.3.1.Visual Studio Code

Kod za web aplikaciju je izrađen u Visual Studio Code. Taj uređivač koda je izabran zbog mogućnosti korištenja terminala unutar uređivača koda te dostupnih ekstenzija od kojih su korišteni Live Server, HTML Snippets i HTML CSS Support. Live Server je korišten kako bi se kod mogao testirati lokalno i izbjegao relativno dug postupak objavljivanja koda na Firebase Hosting koji traje samo nekoliko sekundi, ali bi svakako predstavljao prepreku kada bi se morao pokretati nakon svake promjene u kodu. Druge dvije ekstenzije su korištene zbog mogućnosti bojanja koda po sintaksi za jednostavnije pisanje koda. Korištenje terminala unutar uređivača koda je bilo potrebno za korištenje Webpack i Firebase CLI.

2.3.2.Arduino IDE

Kako bi postavili kod na Arduino mikro kontroler za izvršavanje, najjednostavniji i predviđeni način za to je preko Arduino IDE koji postoji u online i offline verzijama.

Kako bi postavili kod na Arduino mikro kontroler za izvršavanje, korišten je Arduino IDE zbog jednostavnijeg kompiliranja i postavljanja koda te instaliranja biblioteka koje su potrebne za korištenje WiFi mogućnosti te dohvaćanja mjerenja senzora.

2.4. Tehnologije

2.4.1. Arduino biblioteke

Kako bi Arduino uspio komunicirati sa senzorima, bazom podataka i dohvaćao trenutno vrijeme, potrebne su mu biblioteke nakon čijeg instaliranja i uključivanja u kod naredbom *#include* je moguće koristiti funkcije za navedene potrebe. Biblioteke koje su korištene u ovome radu su:

- *Firebase Arduino based on WiFiNINA*
- *UnixTime*
- *Arduino_Sensorkit*

Firebase Arduino biblioteka je korištena kako bi uspjeli slati mjerenja koja zabilježimo na *Firebase* bazu podataka, a *UnixTime* je korišten kako bi uspjeli dohvatiti trenutno vrijeme. *Arduino_Sensorkit* je standardna biblioteka za korištenje s prethodno navedenim *Sensor kit*.

2.4.2. Node.js

Node.js je potreban zbog njegovog pratećeg menadžera paketa, npm. Npm je menadžer paketa pomoću kojeg instaliramo *Firebase* naredbom *npm install firebase* i pokrećemo webpack.

2.4.3. Webpack

Webpack je *module bundler* te je korišten zato što novije verzije Firebase usluge (verzije 9.0 i novije) su optimizirane za rad preko modula i koristeći *module bundlere* poput webpack, fusebox, rollup i drugih.

2.4.4. Chart.js

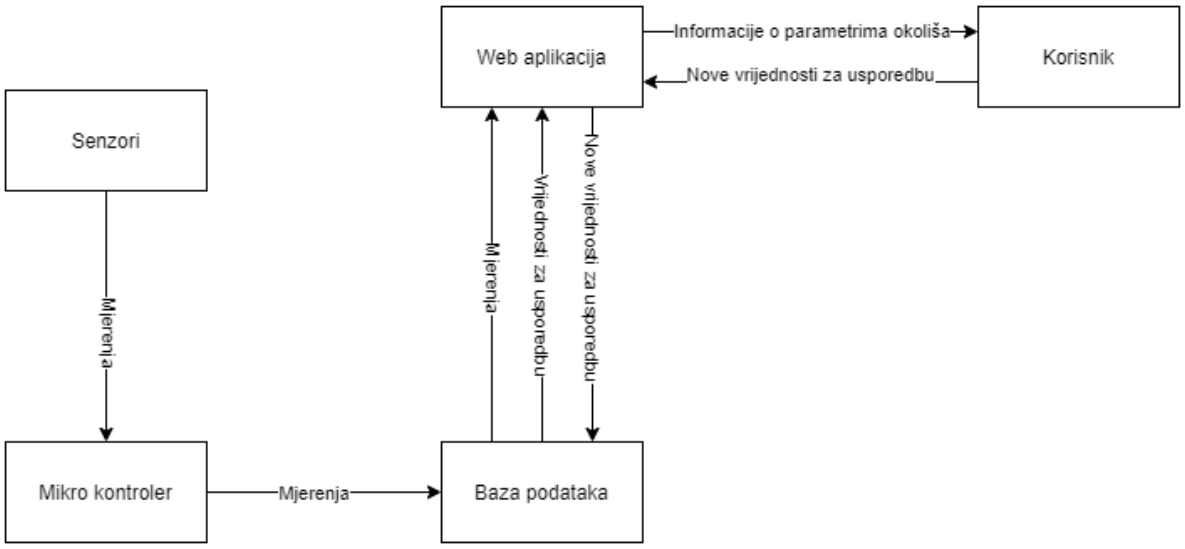
U svrhu jednostavnijeg prikazivanja mjerenja na web aplikaciji, odlučeno je prikazati ih preko individualnih grafova. Grafovi su nacrtani koristeći Graph.js biblioteku jer je izdašno objašnjeno korištenje uputama te primjerima raznih tipova grafova.

3. Razrada teme

Glavni dio ovog rada se sastoji od 4 dijela. U prvome dijelu će biti prikazan cjelokupni sustav i njegovi pojedini dijelovi te kako funkcioniraju zajedno i međusobno surađuju. Poslije toga će biti prikazani pojedini senzori i kako uspijevaju pomoću kojih principa i zakona fizike čitati parametre okoliša. Nakon toga će biti prikazan kod za senzore kojim mikro kontroler dohvaća mjerenja, kako dohvaća trenutno vrijeme i kako šalje podatke na bazu podataka. U posljednjem dijelu će biti obrađena web aplikacija, kako ona dohvaća mjerenja iz baze podataka, crta grafove s dohvaćenim podatcima, kako funkcionira ikona te kako se ažuriraju vrijednosti za upozoravanje ikonom.

3.1. Rad sustava

U ovome poglavlju će biti prikazan cjelokupni sustav pojednostavljenim dijagramom prema kojem će biti objašnjeno njegovo funkcioniranje.



Slika 1: Dijagram rada sustava[autorski rad]

Kao prvi korak u sustavu, mikro kontroler svake minute preko senzora prikuplja mjerenja okoliša koja senzori izmjere. Nakon toga ih šalje na bazu podataka preko WiFi veze na koju se spoji pri pokretanju i ostaje spojen tijekom rada.

Nakon toga, mjerenja su spremljena u bazu podataka i mogu se prikazivati na web aplikaciji. Kada korisnik pristupi web aplikaciji, dohvaćaju se mjerenja iz baze podataka i

prezentiraju korisniku na zadani način preko grafova i ikona. Ako je posljednje izmjerena vrijednost nekog parametra veća od posljednje zadane vrijednosti za upozoravanje za taj parametar, pozadina ikone u odsječku tog parametra će se promijeniti u crvenu iz zelene.

Ako korisnik želi promijeniti vrijednosti za upozoravanje, to može uraditi preko web aplikacije koja nove vrijednosti sprema u bazu podataka preko starih odakle ih povlači odmah i ažurira ikone i pri svakom sljedećem pristupanju.

3.2. Senzori

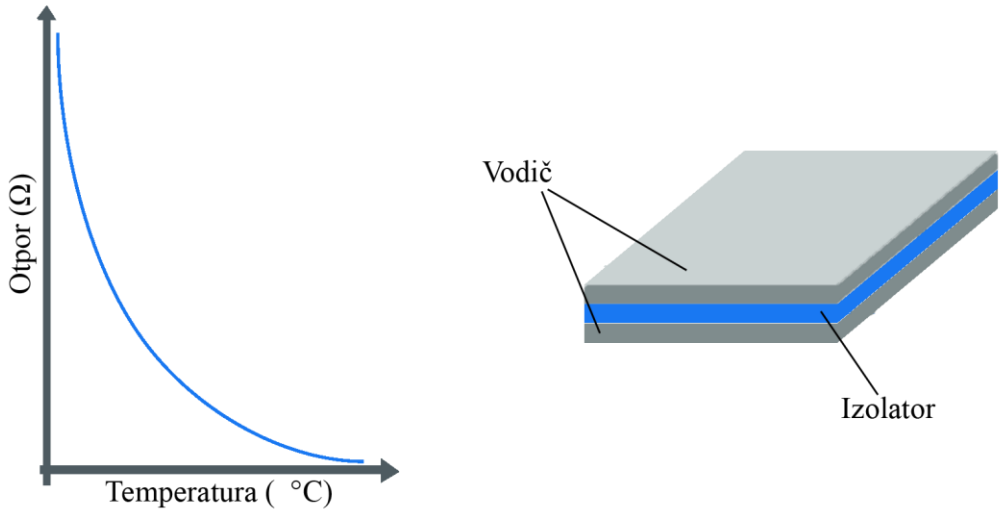
U ovome poglavlju ćemo proći kako pojedini senzori funkcioniraju i prikazati njihovu internu strukturu. Objašnjenja će biti rudimentarna zbog fokusa ovog rada na dio koji je potrebno samostalno uraditi i na razvoju usluge.

3.2.1. Senzor za temperaturu i vlagu

Prvi od senzora koji ćemo obraditi jeste senzor za temperaturu i vlagu. Korišteni senzor je DHT20 senzor koji se sastoji od 2 dijela.

Prvi dio je senzor koji mjeri vlagu i to je kondenzator koji ima dvije ploče i izolator između njih. Mjerenje se ostvaruje tako što se mijenja mjerena voltaža nakon što se promijeni kapacitet kondenzatora upijanjem vlage.[1]

Drugi senzor također pruža rezultat na osnovu voltaže. Ta voltaža se mijenja u termistoru koji sačinjava taj senzor tako što se povećanjem temperature smanjuje otpor.[1]

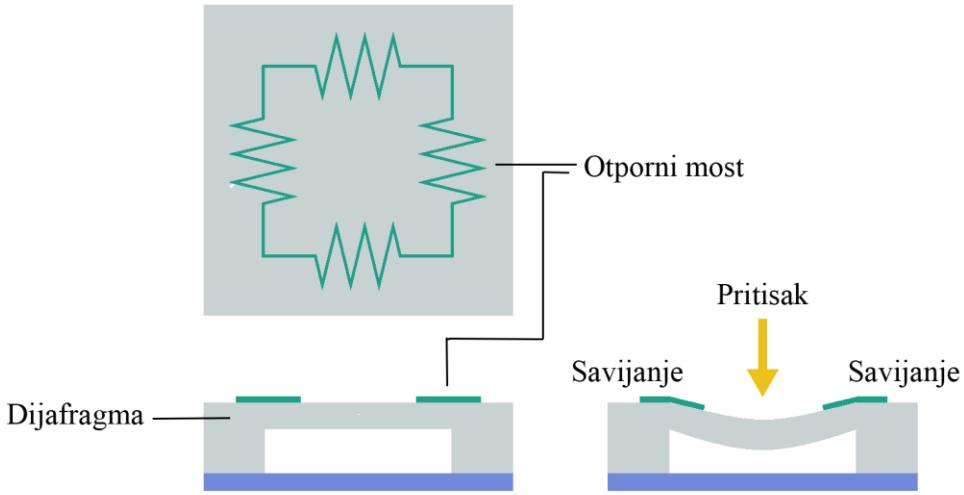


Slika 2: Graf ovisnosti otpora o temperaturi i skica kondenzatora (Prema: sensorkit.arduino.cc)

3.2.2.Senzor za pritisak

Pritisak zraka mjeri se barometrom, a korišteni barometar je Bosch BMP280 na Grove barometar senzoru. Navedeni senzor mjeri podatke o izmjerenom pritisku i temperaturi barometrom i preko I2C protokola ih prosljeđuje mikro kontroleru.[2]

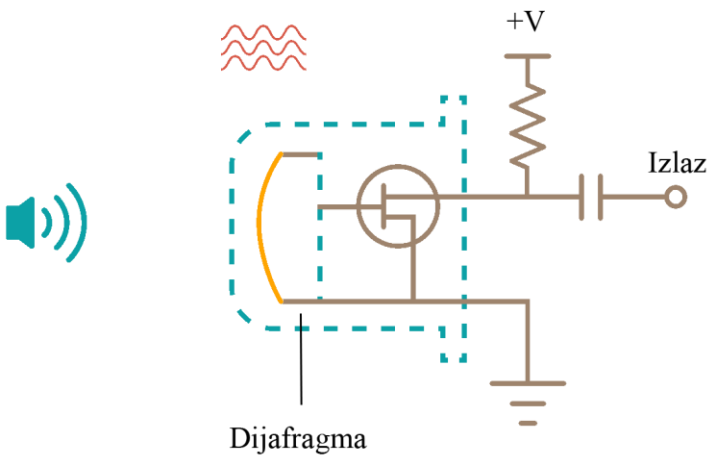
Pomoću ovog senzora je također moguće izmjeriti i temperaturu umjesto prethodno obrađenog senzora, ali je izabran prethodni senzor za tu funkciju kako mu je to jedna od glavnih funkcija uz mjerenje vlage. Također je moguće mjeriti i nadmorsku visinu što nije uključeno zbog namjere senzora da ne mijenja lokaciju.



Slika 3: Skica funkcioniranja barometra (Prema: sensorkit.arduino.cc)

3.2.3.Senzor za zvuk

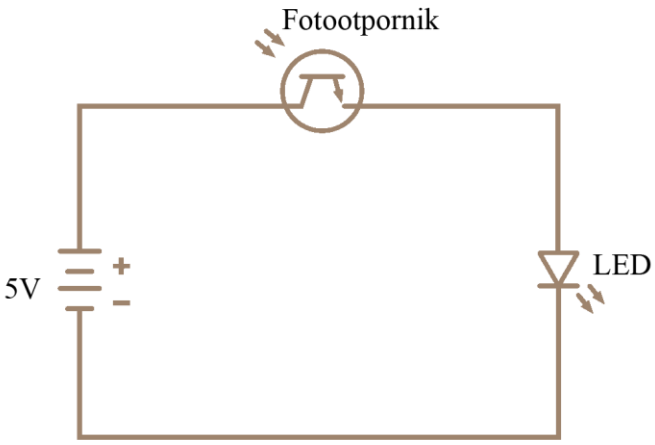
Glavni dio senzora za zvuk jeste dijafragma. Zvukovi svojom vibracijom uzrokuju da dijafragma vibrira što mijenja njen električni kapacitet i time voltažu. Što je neki zvuk glasniji, time su i njegove vibracije jače. Jače vibracije će imati veći utjecaj na dijafragmu i time na izmjerenu voltažu.[3] Voltaža se, nakon što se izmjeri, prikazuje kao vrijednost između 0 i 1023 te kodom mijenja na vrijednost između 0 i 100 koja se prikazuje.



Slika 4: Skica funkcioniranja senzora za zvuk (Prema: sensorkit.arduino.cc)

3.2.4.Senzor za svjetlost

Senzor za svjetlost funkcionira na sličan način kao i senzor za zvuk. Razlika je u tome da senzor za svjetlost koristi foto otpornik gdje senzor za zvuk koristi dijafragmu i u senzoru za svjetlost se mijenja otpor foto otpornika tako što je otpor manji što je jača svjetlost kojoj je izložen i time je veća mjerena voltaža.[4]



Slika 5: Skica funkcioniranja senzora za svjetlost (Prema: sensorkit.arduino.cc)

3.3. Mjerenje parametara okoliša

Nakon što smo obradili kako senzori prikupljaju mjerenja iz okoliša, proći ćemo kroz kod koji koristi mikro kontroler. Ovaj kod je zadužen za pokretanje senzora, spajanje na WiFi i za slanje podataka na bazu podataka te se nalazi na mikro kontroleru i pokreće svaki put kada se pokrene mikro kontroler.

Prvi korak je uključiti sve potrebne biblioteke. To su: biblioteka za komunikaciju s bazom, biblioteka za dohvaćanje vremena, dvije biblioteke za komunikaciju sa senzorima.

```
#include <Firebase_Arduino_WiFiNINA.h>
#include <UnixTime.h>
#include "Arduino_SensorKit.h"
#include "Wire.h"
```

Dalje je potrebno definirati određene konstante kako bi biblioteke mogle funkcionirati ispravno. Prve 4 konstante su potrebne za komunikaciju s bazom podataka, prve dvije od njih definiraju bazu i omogućuju pristup, a druge dvije su potrebne kako bi se spojili na WiFi mrežu dok je posljednja potrebna senzoru za temperaturu i vlagu.

```
#define FIREBASE_HOST "URL baze podataka"
#define FIREBASE_AUTH "Database secret token"
#define WIFI_SSID "naziv WiFi mreže"
#define WIFI_PASSWORD "Lozinka Wifi mreže"
#define Environment Environment_I2C
```

Posljednji korak u *loop* funkciji kako bi mogli početi izvršavati glavni dio koda jeste povezati se na WiFi mrežu koju smo definirali prethodno za što program ulazi u petlju dok se ne poveže.

```
Serial.print("Connecting to WiFi...");
int status = WL_IDLE_STATUS;
while (status != WL_CONNECTED) {
  status = WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print(".");
  delay(300);
}
Serial.print(" IP: ");
Serial.println(WiFi.localIP());
Serial.println();
```

3.3.1. Mjerenje temperature i vlage

Do verzije 1.0.10 *Arduino_Sensorkit* biblioteke je postojao problem gdje kompleti s DHT20 senzorom za temperaturu i vlagu umjesto DHT11 senzora nisu mogli dobivati mjerenja na predviđeni način nego su dobivali *NaN (Not a number)* očitavanja od senzora. S time je bilo potrebno koristiti drugi pristup. To je urađeno tako što su korištene druge biblioteke i naredbe. Prvo, potrebne biblioteke su uključene prije *setup* funkcije naredbama:

```
#include "Wire.h"
#include "DHT.h"
#define DHTTYPE DHT20
DHT dht(DHTTYPE);
```

A kako bi dobili mjerenja, deklarirali smo tri varijable, 2 *float* varijable i jedno polje s dvije *float* vrijednosti. Mjerenja se prvo spremaju u niz prije nego ih spremimo u zasebne varijable.

```
float humidityMeasurement;
float temperatureMeasurement;
float temp_hum_val[2] = {0};

if (!dht.readTempAndHumidity(temp_hum_val)) {
  humidityMeasurement = temp_hum_val[0];
  temperatureMeasurement = temp_hum_val[1];
} else {
  Serial.println("Failed to get temprature and humidity value.");
}
```

Nakon dolaska verzije 1.0.10, mjerenja je moguće dobiti na mnogo jednostavniji način. Prvo je potrebno inicijalizirati biblioteke što se obavlja funkcijama

```
Wire.begin();
Environment.begin();
```

Sva mjerenja i trenutno vrijeme se dobiva preko zasebnih funkcija, a temperatura se dobiva preko *getTemperature* funkcije koja ne prima ulazne argumente i vrača jedan argument tipa *float* koji jeste varijabla „*temperature*“. U tu varijablu dobivamo vrijednost pomoću funkcije *Environment.readTemperature.*

```
float getTemperature(){
  float temperature;
  temperature = Environment.readTemperature();
  return temperature;
}
```

Temperatura i vlaga su karakteristične po tome što oba mjerenja dobivamo s istog senzora te tako koriste funkcije iz iste biblioteke, tj. *Environment* biblioteke. Vlaga se također

nalazi u vlastitoj funkciji koja je strukturno identična funkciji za temperaturu s promjenama samo u nazivima funkcija i varijabli.

```
float getHumidity(){
  float humidity;
  humidity = Environment.readHumidity();
  return humidity;
}
```

Pošto smo funkcije odvojili van glavne *loop* funkcije, moramo ih zasebno pozvati u *loop* funkciji i spremiti vrijednosti koje vrate u zasebne varijable.

```
float temperatureMeasurement = getTemperature();
float humidityMeasurement = getHumidity();
```

3.3.2. Mjerenje pritiska

Pritisak, za razliku od temperature i vlage, ne dobivamo preko funkcija iz *Environment* biblioteke nego iz *Pressure* biblioteke te je s time moramo i inicijalizirati u *setup* funkciji naredbom:

```
Pressure.begin();
```

Nakon toga, ponovno u zasebnoj funkciji dobivamo željena mjerenja na strukturno identičan način kao i prethodna dva mjerenja.

```
float getPressure(){
  float pressure;
  pressure = Pressure.readPressure();
  return pressure;
}
```

I konačno, u *loop* dijelu programa moramo pozvati funkciju i spremiti je u varijablu.

```
float pressureMeasurement = getPressure();
```

3.3.3. Mjerenje zvuka

Senzori za zvuk i svjetlost se razlikuju od prethodna dva senzora po tome da je potrebno deklarirati na koje iglice su priključeni senzori. Kako su senzori korišteni odmah s ploče, nalaze se na zadanim iglicama što je za senzor zvuka A2.

```
int sound_sensor = A2;
```

Sami način dobivanja mjerenja je nalik prethodnima, preko zasebnih funkcija čije povratne vrijednosti spremamo u varijable. Mjerenja buke i svjetlosti spremamo u *int* varijable.

```
int soundMeasurement = getSound();
```

Funkcije također imaju i jedan dodatni dio gdje vrijednosti mijenjamo iz raspona od 0 do 1023 na raspon između 1 i 100 radi jednostavnosti.

```
int getSound(){

  int rawSound = 0;
  for (int i = 0; i < 32; i++){
    rawSound += analogRead(sound_sensor);
  }
  rawSound >>= 5; //bitshift operation

  int soundValue = map(rawSound, 0, 1023, 0, 100); // map the value from 0, 1023 to 0, 100
  return soundValue;
}
```

3.3.4. Mjerenje svjetlosti

Senzor svjetlosti također treba deklarirati na kojoj iglici se nalazi, a standardno se nalazi odmah do senzora za zvuk na iglici A3.

```
int light_sensor = A3;
```

Dohvaćanje mjerenja je identično kao i za mjerenje zvuka.

```
  int lightMeasurement = getLight();
```

A sama funkcija je jednaka funkciji za dohvaćanje zvuka bez petlje i pomjeranja bitova koji su potrebni za dohvaćanje zvuka.

```
int getLight(){
  int raw_light = analogRead(light_sensor); // read the raw value from light_sensor pin (A3)
  int light = map(raw_light, 0, 1023, 0, 100); // map the value from 0, 1023 to 0, 100
  return light;
}
```

3.3.5. Dohvaćanje trenutnog vremena

Dodatna funkcija koju sustav obavlja je dohvaćanje vremena. Za to prvo postavljamo vremensku zonu u kojoj se nalazimo.

```
UnixTime stamp(2);
```

Zatim vrijeme dohvaćamo preko zasebne funkcije bez ulaznih parametara i s jednim povratnim argumentom tipa *String*. U funkciji prvo dohvaćamo vrijeme s mikro kontrolera i zatim ga prebacujemo u traženi format. Također dodajemo 0 ispred jednoznamenkastih

mjeseci, dana, sati i minuta kako bi se unosi pravilno spremali u bazu podataka jer datum koristimo kao ključ za unose. Također, zbog pravila baze podataka, u datumu je korištena povlaka umjesto točke jer točka nije dozvoljena u nazivu ključeva.

```
String getDateAndTime(){

  uint32_t unixTime = WiFi.getTime();
  stamp.getDateTime(unixTime);

  String currentDateAndTime = "";

  //currentDateAndTime = currentDateAndTime + "/"
  currentDateAndTime = currentDateAndTime + stamp.year;
  currentDateAndTime = currentDateAndTime + "-";
  if (stamp.month < 10){
    currentDateAndTime = currentDateAndTime + "0";
    currentDateAndTime = currentDateAndTime + stamp.month;
  }else{
    currentDateAndTime = currentDateAndTime + stamp.month;
  }
  currentDateAndTime = currentDateAndTime + "-";
  if(stamp.day < 10){
    currentDateAndTime = currentDateAndTime + "0";
    currentDateAndTime = currentDateAndTime + stamp.day;
  }else{
    currentDateAndTime = currentDateAndTime + stamp.day;
  }
  currentDateAndTime = currentDateAndTime + " ";
  if(stamp.hour < 10){
    currentDateAndTime = currentDateAndTime + "0";
    currentDateAndTime = currentDateAndTime + stamp.hour;
  }else{
    currentDateAndTime = currentDateAndTime + stamp.hour;
  }
  currentDateAndTime = currentDateAndTime + ":";
  if(stamp.minute < 10){
    currentDateAndTime = currentDateAndTime + "0";
    currentDateAndTime = currentDateAndTime + stamp.minute;
  }else{
    currentDateAndTime = currentDateAndTime + stamp.minute;
  }
```

```
  return currentDateAndTime;
}
```

I vrijeme također dohvaćamo u loop funkciji i spremamo u varijablu.

```
  String dateAndTimeOfMeasurement = getDateAndTime();
```

3.3.6. Slanje mjerenja

Nakon što prikupimo sva mjerenja i trenutni datum, možemo poslati podatke na bazu podataka što radimo pomoću prethodno navedene biblioteke nakon što prvotno deklariramo potrebne varijable. Varijabla *path* nam označava u koji čvor želimo spremati podatke, a varijablom *pathNew* dodajemo datum u putanju kako bi ga mogli koristiti kao ključ.

```
FirebaseData firebaseData;
String path = "/measurements";
String pathNew;
```

Potrebne argumente za naredbu *Firebase.begin* moramo također definirati svaki individualno pomoću *#define*. Iz razloga privatnosti i sigurnosti, deklaracije nisu prikazane u ovome radu. To su:

- FIREBASE_HOST - URL baze podataka bez „https://“
- FIREBASE_AUTH – Token „*database secret*“
- WIFI_SSID - Naziv WiFi mreže na koju se povezujemo
- WIFI_PASSWORD - Lozinka WiFi mreže na koju se povezujemo

```
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH, WIFI_SSID, WIFI_PASSWORD);
```

Nakon što smo deklarirali varijablu za putanju, potrebno je pridodati joj datum i vrijeme kako bi se izmjerene vrijednosti spremile na željeno mjesto i zadržali strukturu baze podataka.

```
pathNew = "";
pathNew = path + "/";
pathNew = pathNew + dateAndTimeOfMeasurement;
```

Podatke je moguće zatim, nakon što obavimo neophodne pripreme, poslati na bazu podataka naredbom *Firebase.setFloat* te također isprintamo sva mjerenja u *Serial Monitor*.

```
if (Firebase.setFloat(firebaseData, pathNew + "/light", lightMeasurement))
{
      Serial.println(firebaseData.dataPath() + " = " + lightMeasurement);
}
if (Firebase.setFloat(firebaseData, pathNew + "/sound", soundMeasurement))
{
      Serial.println(firebaseData.dataPath() + " = " + soundMeasurement);
}
```

```
if (Firebase.setFloat(firebaseData, pathNew + "/pressure",
pressureMeasurement)) {
      Serial.println(firebaseData.dataPath() + " = " +
pressureMeasurement);
}
if (Firebase.setFloat(firebaseData, pathNew + "/humidity",
humidityMeasurement)) {
      Serial.println(firebaseData.dataPath() + " = " +
humidityMeasurement);
}
if (Firebase.setFloat(firebaseData, pathNew + "/temperature",
temperatureMeasurement)) {
      Serial.println(firebaseData.dataPath() + " = " +
temperatureMeasurement);
}
```

3.4. Web aplikacija

Kako bi korisnik mogao jednostavnije provjeravati parametre okoliša, napravljena je i web aplikacija preko koje je moguće pogledati sve parametre pojedinačno te njihovih posljednjih 10 mjerenja preko grafa. S lijeve strane se nalaze ikone s prikazima parametara u zelenom krugu ako je posljednja izmjerena vrijednost manja od zadane vrijednosti za usporedbu, a crvena ako je vrijednost veća kako bi upozorilo na visoko mjerenje. Zadanu vrijednost za usporedbu je moguće promijeniti tako da u odjeljenju jednog parametra, tj. ispod ikone parametra, u polje za unos unesemo novu brojčanu vrijednost koju želimo za taj parametar i kliknemo na gumb pored. Moguće je mijenjati po jedan parametar ili više njih u istom trenutku tako što unesemo nove vrijednosti u više polja za unos i kliknemo na bilo koji gumb. Ako ne želimo promijeniti neku vrijednost, polje ostavimo prazno i vrijednost za taj parametar će ostati nepromijenjena.

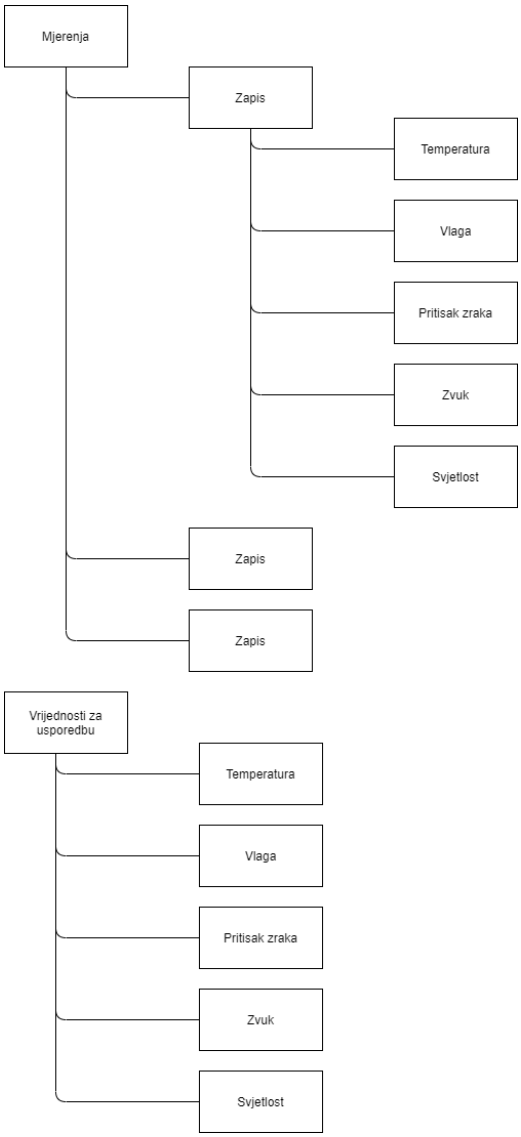
Slika 6: Prikaz web aplikacije [autorski rad]

HTML kod za stranicu se naravno sastoji od svih dijelova koji su potrebni za funkcioniranje stranice, a posebno ćemo obraditi samo glavni dio. Svi grafovi i ikone se nalaze u jednom div odjeljku klase *container*. U tom odjeljku se nalazi pet identičnih odjeljaka klase „part“. Svaki odjeljak „part“ se dalje dijeli na dva dijela klasa *icon-form* i *canvas*. *Canvas* odjeljak sadrži graf, a odjeljak *icon-form* sadrži dijelove *icon* u kojem je ikona i *form* u kojem su polje za unos i gumb. Struktura je identična za svih 5 parametara, a različiti su samo nazivi klasa, *canvasa*, *id* i ikona.

```
<div class="part">
  <div class="icon-form">
    <div class="icon" id="temperatureIDicon">
      <i class="fa-solid fa-temperature-half"></i>
    </div>
    <div class="form">
      <input type="text" placeholder="Input field">
      <button id="temperatureIDbutton">Submit</button>
    </div>
  </div>
  <div class="canvas">
    <canvas id="temperature (°C)"></canvas>
  </div>
</div>
```

3.4.1. Struktura baze podataka

Firebase Realtime Database se bazira na NoSQL principu što znači da ne koristi relacione tablice sa slogovima koji imaju određene atribute nego JSON zapis. Baza podataka za ovaj projekt ima 2 glavna dijela. Prvi čvor sadrži niz zapisa koji sadrže vrijednosti senzora za sve mjerene parametre, a zapis se naziva prema vremenu u kojem je izmjereno. Drugi dio jeste jedan zapis na prvoj razini, tj. van prvog dijela, u kojem se nalazi po jedan zapis za svaki od mjerenih parametara koje zadaje korisnik i s kojima se uspoređuje posljednja izmjerena vrijednost te koji se ažuriraju kada korisnik promjeni vrijednost.



Slika 7: Prikaz strukture baze podataka [autorski rad]

3.4.2.Firebase config objekt

Od verzije 9, Google Firebase uvodi podršku i prebacuje fokus na modularni način korištenja usluge zbog manje veličine SDK i bolje učinkovitosti kada se koriste alati poput Webpack koji je korišten u ovome projektu.[5]

Na početku koda uvezemo potrebne funkcije koje ćemo koristiti iz njihovih zasebnih biblioteka.

```
import { initializeApp } from 'firebase/app'
import { getDatabase, get, ref, limitToLast, query, set, update } from
"firebase/database";
```

Zatim obavezno moramo dodati *firebaseConfig* objekt koji je jedinstven za projekt koji prethodno kreiramo u Google Firebase konzoli i sadrži jedinstvene podatke za taj projekt koji nisu tajni te ih stoga upisujemo u kod web aplikacije. Navedeni objekt je moguće u svakom trenutku preuzeti s Firebase konzole i uključiti u kod projekta. *Config* objekt se koristi za web aplikacije dok se za Android aplikacije koriste JSON datoteka i PLIST datoteka za Apple aplikacije. Svaka od tri navedene opcije služe da povežu aplikaciju s Firebase projektom i njegovim resursima poput baze podataka koja je korištena u ovom projektu.[6]

```
const firebaseConfig = {
    apiKey: "AIzaSyBG4P83mYO2oJiAjnkC-bPQz3NnDQoAUyk",
    authDomain: "scecura-zavrsnirad.firebaseapp.com",
    databaseURL: "https://scecura-zavrsnirad-default-rtdb.europe-
west1.firebasedatabase.app",
    projectId: "scecura-zavrsnirad",
    storageBucket: "scecura-zavrsnirad.appspot.com",
    messagingSenderId: "98494566087",
    appId: "1:98494566087:web:498a504ea2d1d90959f8e2"
  };
```

Kao posljednji korak u pripremi web aplikacije je potrebno inicijalizirati aplikaciju i dohvatiti instancu baze podataka funkcijama koje smo uključili prethodno.

```
const app = initializeApp(firebaseConfig);
const database = getDatabase(app);
```

3.4.3. Dohvaćanje mjerenja

Kako bismo kasnije u kodu mogli koristiti mjerenja za crtanje grafova i upozoravanje na visoka mjerenja, mjerenja ćemo spremiti u zasebne nizove koje prvo kreiramo van svih funkcija kako bi ih mogli koristiti u više njih.

```
const temperatureArray = [];
const humidityArray = [];
const pressureArray = [];
const lightArray = [];
```

```
const soundArray = [];
```

Prvo ćemo napraviti upit pomoću kojeg ćemo dohvatiti željena mjerenja tako da mu proslijedimo kao argument instancu baze podataka i lokaciju unutar baze podataka s koje želimo dohvatiti podatke, a kao drugi argument funkciji *query* ćemo dodati argument *limitToLast* s vrijednošću 10 kako bi dohvatili samo posljednjih 10 mjerenja. Drugi argument je dodan kako ne bi svaki put dohvaćali sva mjerenja nego samo posljednjih 10 radi brzine učitavanja i smanjene potrošnje resursa.

Nakon što kreiramo upit ćemo dohvatiti željene podatke, odmah ćemo nastaviti funkciju i unutar nje iterirati po svim čvorovima dohvaćenog objekta i svaki od njih proslijediti u funkciju *arrayAdding*.

```
const databaseref = query(ref(database, `/measurements`), limitToLast(10));
get(databaseref).then((snapshot) => {

  snapshot.forEach(childSnapshot => {
    arrayAdding(childSnapshot);
  })

});
```

Unutar funkcije *arrayAdding* uzimamo vrijednosti atributa dobivenog objekta koji su zapravo mjerenja koja su spremljena u istome trenutku i spremamo ih na prvo prazno mjesto u prethodno deklariranim nizovima. Svaki od mjerenih parametara se sprema u zaseban niz i sva mjerenja se spremaju tako da svaki zapis u nizu sadrži dva argumenta, datum zapisa i vrijednost parametra.

```
function arrayAdding(childSnapshot){

  temperatureArray.push({dateAndTime: childSnapshot.key, temperature:
childSnapshot.val().temperature});
  humidityArray.push({dateAndTime: childSnapshot.key, humidity:
childSnapshot.val().humidity});
  pressureArray.push({dateAndTime: childSnapshot.key, pressure:
childSnapshot.val().pressure});
  soundArray.push({dateAndTime: childSnapshot.key, sound:
childSnapshot.val().sound});
  lightArray.push({dateAndTime: childSnapshot.key, light:
childSnapshot.val().light});

}
```

3.4.4.Crtanje grafova

Nakon što izađemo iz prethodne petlje i još uvijek smo u funkciji u kojoj smo dohvatili posljednjih 10 mjerenja, za svaki od mjerenih parametara pozivamo funkciju *drawChart* kojoj prosljeđujemo: naziv platna u koji želimo nacrtati graf, datume posljednjih 10 mjerenja i posljednjih 10 izmjerenih vrijednosti tog parametra. Datume i mjerenja prosljeđujemo iz nizova koje smo prethodno kreirali.

```
  drawChart("temperature (°C)", temperatureArray.map(obj =>
obj.dateAndTime), temperatureArray.map(obj => obj.temperature));

  drawChart("humidity (%)", humidityArray.map(obj => obj.dateAndTime),
humidityArray.map(obj => obj.humidity));

  drawChart("pressure (Pa)", pressureArray.map(obj => obj.dateAndTime),
pressureArray.map(obj => obj.pressure));

  drawChart("light (0-100)", lightArray.map(obj => obj.dateAndTime),
lightArray.map(obj => obj.light));

  drawChart("sound (0-100)", soundArray.map(obj => obj.dateAndTime),
soundArray.map(obj => obj.sound));
```

Funkcija *drawChart* crta graf preko Chart.js biblioteke tako da napravi novi objekt u kojem definiramo par vrijednosti preko argumenata koje proslijedimo. Ime platna u kojem će se graf nalaziti dobivamo preko *canvasid*, vrijednosti na x osovini su datumi posljednjih 10 mjerenja i nalaze se u nizu *labels*, a vrijednosti mjerenja su u nizu *measurements.* Za argument *type* je korišteno *'line'* jer linijski graf ima najviše smisla za željeni učinak zbog mogućnosti pregleda trenda parametra, u *datasets* se nalaze *canvasid* za naziv iznad grafa što je razlog zašto naziv sadrži i mjerne jedinice i *measurements,* a *labels* sadrži istoimenu varijablu. Biblioteka automatski popunjava vrijednosti na y osi, a točne vrijednosti pojedinih mjerenja je moguće vidjeti tako da pokazivač stavimo iznad točke mjerenja.

```
function drawChart(canvasid, labels, measurements){


  new Chart(
    document.getElementById(canvasid),
    {
      type: 'line',
      data: {
        datasets: [
          {
            label: canvasid,
            data: measurements
          }
        ],
        labels: labels
      }
    }
```

```
  );
}
```

3.4.5.Ikona mjerenja

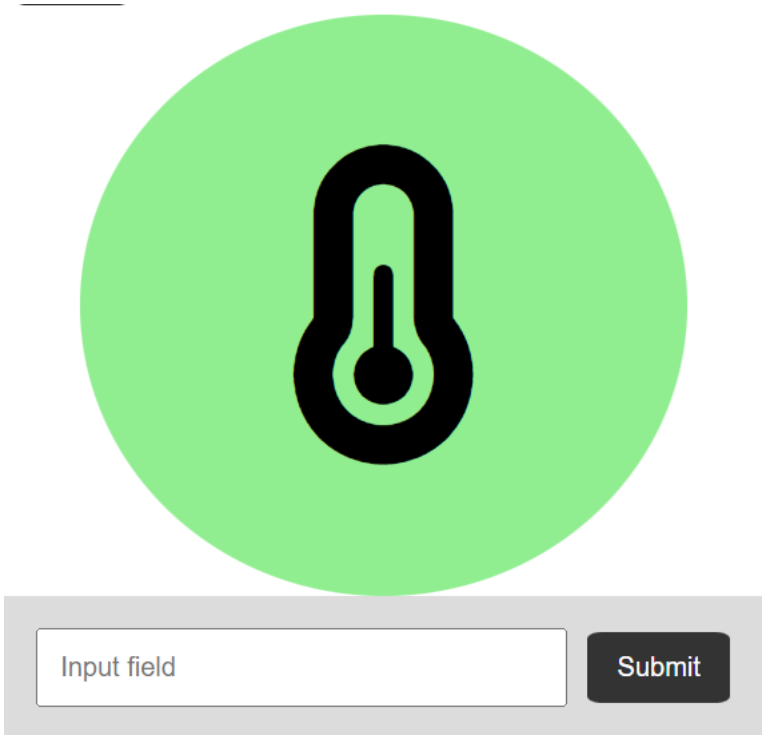
Kako bismo mogli usporediti posljednje izmjerenu vrijednost parametra sa zadanim vrijednostima iznad kojih treba upozoriti na visoko mjerenje, moramo ih prvo dohvatiti što radimo na identičan način kako smo dohvatili i mjerenja, tj. preko upita i instance baze podataka. Nakon što dohvatimo vrijednosti za usporedbu, proslijedit ćemo ih u funkciju *valueCheck*. Prvi argument je posljednja izmjerena vrijednost, drugi argument je vrijednost s kojom uspoređujemo, a posljednji argument jeste *id* odjeljenja u kojem se nalazi ikona za taj parametar. Funkcija se poziva 5 puta, jednom za svaki parametar koji mjerimo s odgovarajućim argumentima za parametar za koji se poziva.

```
const comparisonRef = query(ref(database, `/comparison`));
get(comparisonRef).then((snapshot) => {


  valueCheck(temperatureArray[temperatureArray.length - 1].temperature,
snapshot.val().temperature, "temperatureIDicon");
  valueCheck(humidityArray[humidityArray.length - 1].humidity,
snapshot.val().humidity, "humidityIDicon");
  valueCheck(pressureArray[pressureArray.length - 1].pressure,
snapshot.val().pressure, "pressureIDicon");
  valueCheck(soundArray[soundArray.length - 1].sound, snapshot.val().sound,
"soundIDicon");
  valueCheck(lightArray[lightArray.length - 1].light, snapshot.val().light,
"lightIDicon");


});
```

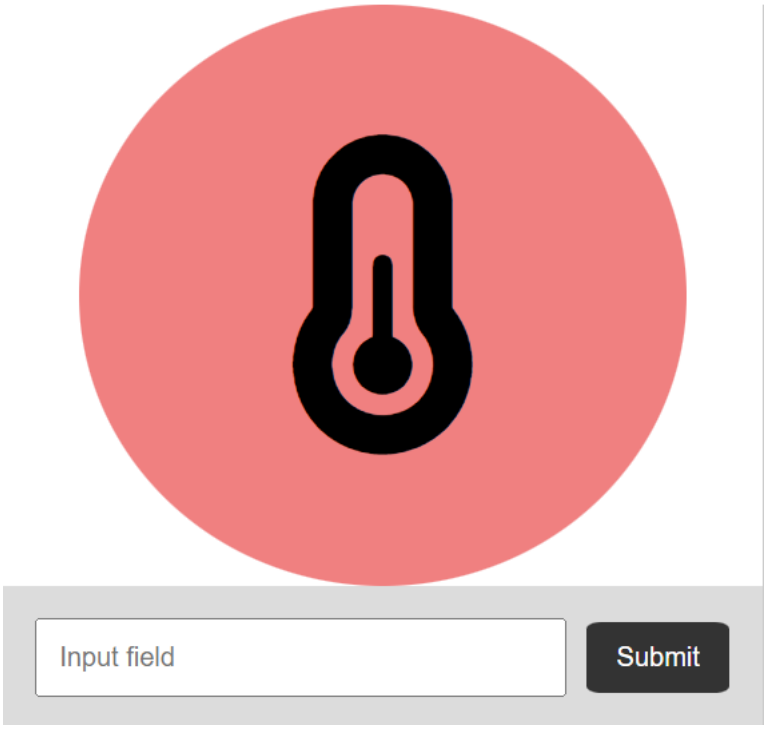
Sama funkcija *valueCheck* je vrlo jednostavna jer se sastoji od jednog *if* grananja u kojem pozadinu ikone mijenjamo u svjetlo crvenu, tj. *lightcoral*, boju ako je posljednja izmjerena vrijednost veća od vrijednosti za usporedbu.

```
function valueCheck(lastOfArray, comparisonVar, iconId){


  const iconElement = document.getElementById(iconId);


  console.log("u value checku", comparisonVar, iconId);


  if (lastOfArray > comparisonVar) {
    iconElement.style.backgroundColor = 'lightcoral';
  }else{
    iconElement.style.backgroundColor = 'lightgreen';
```

```
    }

}
```



Slika 8: Izgled ikone tijekom normalnih vrijednosti [autorski rad]



Slika 9: Izgled ikone s upozorenjem na visoko mjerenje [autorski rad]

3.4.6. Ažuriranje vrijednosti za upozoravanje

Kako bismo uspješno ažurirali vrijednosti za upozoravanje, potrebno je kreirati niz u koji ćemo spremiti sve trenutne vrijednosti iznad kojih web aplikacija upozorava korisnika i to se događa odmah nakon što provjerimo da li najnovija mjerenja premašuju trenutne vrijednost za upozoravanje.

```
comparisonValues = [
    snapshot.val().temperature,
    snapshot.val().humidity,
    snapshot.val().pressure,
    snapshot.val().sound,
    snapshot.val().light
  ]
```

Za svaki od 5 gumbova je dodan *EventListener* koji pokreće funkciju *updateComparisonValues* nakon klika na jedan od gumbova.

```
buttons.forEach(button => {
  button.addEventListener('click', updateComparisonValues);
});
```

U funkciji *updateComparisonValues* se nalaze dva niza, jedan sa svim poljima za unos i drugi u koji ćemo spremati nove vrijednosti za usporedbu za upozoravanje. Prvi korak je proći petljom kroz sva polja za unos i provjeriti da li je u njih unesena vrijednost i spremiti tu vrijednost na odgovarajuće mjesto u nizu *newComparisonValues* ako jest ili vrijednost iz prethodno spomenutog niza comparisonValues ako nije. Drugi korak jeste poslati nove vrijednosti na bazu podataka funkcijom *update*.

```
function updateComparisonValues(event) {

  const newComparisonValues = [];
  const inputFields = document.querySelectorAll('input[type=text]');

  inputFields.forEach((inputField, index) => {

    const inputValue = inputField.value.trim();

    if (inputValue === '') {
      newComparisonValues.push(comparisonValues[index]);
    } else {
      newComparisonValues.push(inputValue);
    }
```

```
  });

  update(ref(database, `/comparison`), {
    temperature: newComparisonValues[0],
    humidity: newComparisonValues[1],
    pressure: newComparisonValues[2],
    sound: newComparisonValues[3],
    light: newComparisonValues[4]
  })

  location.reload();
}
```

4. Zaključak

Koristeći jednostavne i pristupačne alate je napravljena jednostavna web aplikacija preko koje je moguće provjeravati parametre okoliša. Arduino je platforma s velikom raznolikošću mikro kontrolera, senzora i ostalih dijelova koji omogućuju da svaka osoba može napraviti uslugu koja zadovoljava njihove potrebe, a Google Firebase je usluga preko koje je moguće kreirati web stranicu potpuno besplatno. Način korištenja Google Firebase je relativno kompliciran za početnike s obzirom na pristup, ali izobilje uputa čini postupak dostupnim svima dok Arduino sustavi sadrže jednostavne upute i primjere za sve funkcionalnosti dok nedavni trend poboljšanih umjetnih inteligencija omogućava i osobama bez iskustva ili s ograničenim iskustvom da pišu kod. Uređivači koda Visual Studio Code i Arduino IDE su jednostavni za svakog korisnika i sadrže u sebi sve što je potrebno za potpunu izvedbu projekta ovog tipa u potpunosti poput terminala i dodataka unutar VS Code i biblioteka unutar Arduino IDE. Biblioteke za Arduino su vrlo jednostavne za korištenje jer ih je moguće skinuti unutar IDE-a i nude primjere koji imaju komentare i upute.

Prvi dio rada jeste bio opis koje su aplikacije, alati i tehnologije korišteni dok je drugi dio podijeljen na tri dijela. Prvi od ta tri dijela opisuje pojedine senzore koji su korišteni i njihov način funkcioniranja, drugi dio objašnjava Arduino kod i kako su korišteni prethodno obrađeni senzori i u trećem dijelu je objašnjena web aplikacija, prvotno uz njen HTML kod i onda su funkcionalnosti pojedinačno objašnjene uz JavaScript kod.

Uslugu je moguće dodatno poboljšati na par načina:

- Pratiti više parametara,
- Pratiti parametre okoliša na više lokacija,
- Objediniti sva polja za unos i gumbove u jednu formu,
- Upozoravati i na niska mjerenja,

No glavna poenta rada je bila pokazati kako je moguće napraviti cjelokupnu uslugu na jednostavan i pristupačan način te stoga sadrži funkcionalnosti koje su tome prikladne.

Popis literature

[1] Arduino (bez dat.) *THE TEMPERATURE SENSOR* [Na internetu]. Dostupno: https://sensorkit.arduino.cc/sensorkit/module/lessons/lesson/08-the-temperature-sensor [pristupano 8.8.2023.].

[2] Arduino (bez dat.) *THE AIR PRESSURE SENSOR* [Na internetu]. Dostupno: https://sensorkit.arduino.cc/sensorkit/module/lessons/lesson/07-the-air-pressure-sensor [pristupano 8.8.2023.].

[3] Arduino (bez dat.) *THE SOUND SENSOR* [Na internetu]. Dostupno: https://sensorkit.arduino.cc/sensorkit/module/lessons/lesson/06-the-sound-sensor [pristupano 9.8.2023.].

[4] Arduino (bez dat.) *THE LIGHT SENSOR* [Na internetu]. Dostupno: https://sensorkit.arduino.cc/sensorkit/module/lessons/lesson/05-the-light-sensor [pristupano 9.8.2023.].

[5] Google for Developers (bez dat) *Learn more about Web and Firebase* [Na internetu] Dostupno: https://firebase.google.com/docs/web/learn-more [pristupano: 22.8.2023.].

[6] Google for Developers (bez dat.) *Understand Firebase projects* [Na internetu]. Dostupno: https://firebase.google.com/docs/projects/learn-more#config-files-objects [pristupano: 22.8.2023.].

Popis slika