IoT Projekti

Kahvinkeittimen lämpötila-anturi

[IoT-Projekti] Projektityö

Tieto- ja viestintätekniikka insinööri

2023

Jarkko Niemi

Saku Karttunen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintäekniikka, insinööri
Tekijä Jarkko Niemi, Saku Karttunen
Työn nimi IoT Projektityö
Ohjaaja Timo Karppinen

Tiivistelmä
Vuosi 2023

Tämä on raportti IoT Projekti kurssin projektityöstä. Tämän projektin tehtävänä oli luoda IoT ohjain joka auttaisi jossain oikean elämän ongelmassa taikka ideassa. Projektiin oli aikaa neljä viikkoa josta yksi oli hiihtolomaa. Projektimme rakentui ideasta lämpötilaa mittaavasta mikro-ohjaimesta. Tätä mikro-ohjainta voisi sitten käyttää moneen mahdolliseen tehtävään, joista valitsimme että sillä seurattaisiin kahvikoneen kahvin lämpötilaa ja täten kahvin optimaalisen nauttimishetken löytämiseksi.

Rakensimme myös käyttöliittymän jonka avulla mikro-ohjaimesta saatuja arvoja olisi helppo seurata. Halusimme että projektin luomisen aikana pitäisimme mielessä idean että tämä projekti menisi asiakkaalle. Täten päätimme suunnitellä käyttökokemuksen mahdollisimman asiakasystävälliseksi.

Mikro-ohjain on ohjelmoitu Mbed OS:än avulla käyttäen C++ ohjelmointikieltä. Käyttöliittymäksi loimme Web-sovelluksen joka vastaanottaa MQTT dataa ja muuttaa ne helposti luettavaan ja ymmärrettävään muotoon. Tähän käytimme ReactJS sekä Tailwind työkaluja.

Avainsanat IoT, Internet of Things, Mbed OS, C++
Sivut 26 sivua ja liitteitä 1 sivu

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and communication technology, engineer
Author Jarkko Niemi, Saku Karttunen
Subject IoT Projektityö
Supervisors Timo Karppinen

Abstract
Year 2023

This is a report about the IoT Project course's project assignment. For this project, we were tasked to create and develop a IoT micro-controller that would help a hypothetical client with a real world problem. The time for the project was four weeks, of which one of them was Hiihtoloma (Skii-ing holiday). Our project came from the idea of wanting to make a micro-controller for calculating temperature. This micro-controller could be used in many different kinds of cases, of which we chose to use it for reading the temperature of a coffee machine's coffee. This would be for aquiring the optimal time to enjoy your coffee.

We also built an user interface for easily reading the values and data coming from the micro-controller. We wanted to imagine all throughout the project that we would be creating this product for a theoretical customer. With this we wanted to make the user experience as easy and straight-forward as possible.

The micro-controller was programmed using Mbed OS and with the C++ programming language. For the user interface we decided to create and develop a Web application that would receive and read MQTT data and convert them into a readable and easily understandable format. For this we used ReactJS and TailwindCSS as tools.

This report is written completely in Finnish.

Keywords IoT, Internet of Things, Mbed OS, C++
Pages 26 pages and 1 attachment

Liitteet

Liite 1 Mikro-ohjaimen tekniset piirrustukset

1 Johdanto / Tavoitteet

Tämän teknisen raportin tavoitteena on kuvata IOT-projektin suunnittelua ja toteutusta. Projekti koostuu eri komponenttien integroinnista toimivaksi kokonaisuudeksi, jolla on myös selkeä käyttötarkoitus.

Raportti alkaa kertomalla projektin tavoitteista ja siihen liittyvistä teknisistä vaatimuksista. Seuraavaksi esitellään tarvittavaa teoriatietoa, projektiin valittujen laitteiden ominaisuuksia ja näiden toimintaa. Tämän jälkeen tarkastellaan projektin kokonaisuutta ja sen eri osien välisiä yhteyksiä.

Raportissa käsitellään myös projektin ohjelmiston suunnittelua ja toteutusta. Tämä sisältää ohjelmointia, konfigurointia, kalibrointia ja muuta jota tarvitaan projektin toteutuksessa. Näitä asioita käydetään läpi viikottain, ja mahdollisia ongelmia ja niiden ratkaisuja tuodaan esille.

2 Teoria ja suunnitelma

Tässä projektissa hyödynnetään mikrokontrolleria, lämpötila-anturia, WLAN-moduulia ja dataa vastaanottavaa käyttöliittymää.

Tarkoituksena on koostaa näistä toimiva kokonaisuus, joka mahdollistaa reaaliaikaisen lämpötilatiedon keräämisen tietystä kohteesta ja näiden tietojen lähettämisen käyttöliittymään verkkosivulle, joka luodaan tätä projektia varten.

“Tietty kohde” on tässä tapauksessa kahvipannu, jonka lämpötilaa seurataan, jotta tiedetään milloin kahvi on kokonaisuudessaan tippunut kyseiseen pannuun. Kahvin lämpötilakäyrän pitäisi teoriassa tasoittua ja lopettaa nouseminen kun kahvi on lopettanut tippumisen. Tämän avulla pystyisi myös seuraamaan että mikä on optimaalinen aika kahvin nauttimiselle. Esimerkiksi jos haluaa että kahvi istuu hetken kattilassa ennen kuin sen haluaa juoda. Lämpötilaa voisi itse seurata käyrästä ja päättää itse että millä lämpötilalla haluaa nauttia kahvistansa.

3 Laitteet, materiaalit ja menetelmät

Komponentit:

- ST NUCLEO-L432KC
- MLX90614ESF-BAA-000-TU (IR anturi)
- WLAN moduuli (MOD-WIFI-ESP8266)
- Kaksi vastusta (56 kOhm)
- Keraaminen kondensaattori (100 nF)

Mikro-ohjaimen virtalähteenä käytetään USB micro-B kaapelia ja siihen sopivaa seinälaturia joka kytketään verkkovirtaan. Komponentien kytkemiseen maahan ja toisiinsa käytetään tavallisia M-M tai M-F johtoja.

Ohjelmointikielet:

- C++ anturin ohjelmassa, MBED OS
- Web ohjelmointikielet pohjana verkkosovellukselle
- ReactJS ja TailwindCSS työkaluina verkkosovelluksen kehityksessä
- TypeScript ReactJS:än ohjelmointikielenä (.tsx)

Mikro-ohjaimen MBED OS ohjelman rakentamiseen käytetään selainpohjaista Keil Studio Cloudia. Githubia käytetään isännöintialustana koodille, jotta versionhallinta ja yhteistyö helpottuu.

- Käyttöliittymän Github Repository
- Mikro-ohjaimen Github Repository

4 Ratkaisu & Prosessi

4.1 Viikko 1 (13.2. – 17.2.2023)

Viikko 1 oli projektimme aloitus. Olimme kuitenkin aikaisemmassa vaiheessa jo saaneet suunnitelluksi projektimme aiheen ja idean päämäärästä sekä materiaaleista. Olimme koululla

maanantaina sekä tiistaina klo 9.30 – 15.00. Tänä aikaina ehdimme työskentelemään projektimme kanssa paikallisesti ja yhdessä.

Projektin alussa rakensimme testaus ”demo” -ohjelman jolla opettelimme käyttämään anturiamme sekä sen vaatimia kirjastoja. Tämä oli hyvä päätös, sillä anturin saadakseen toimintaan, se vaatikin hieman enemmän kuin aluksi oletimme. Saimme lopulta kuitenkin anturin ja sitä varten suunnitellun ohjelman toimimaan lähes halutulla tavalla. Seuraavaksi käymme hieman läpi ensimmäisen viikon tapahtumia.

4.1.1 Ongelmia

Ensimmäisellä viikolla oli paljon hankaluuksia ja ongelmia projektia aloittaessa, mutta nämä eivät kuitenkaan olleet projektin loppu. Olimme nimittäin varmoja siitä mitä halusimme tuotteemme tekevän ja olimme valmiita opettelemaan sen vaatimia taitoja.

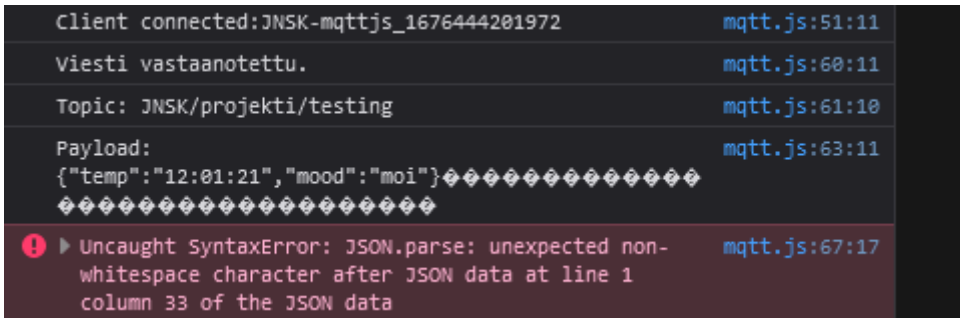
Ensimmäinen ongelma projektin kanssa olikin projektin suurin. Sillä kirjastot joita latasimme Mbed OS sivuilta olivat vanhoja ja rakennettuja Mbed OS 2:n päälle. Kuten voi täten olettaa että kirjastot ja niiden avulla rakennetut ”Hello World” sovellukset eivät siis toimineet. Ehdimme kokeilemaan seuraavat kirjastot, ennen kuin saimme ohjelman toimintaan:

- Hikaru Sugiura:n vuonna 2011 rakennettu MLX90614 -kirjasto.
- Arnas MasRodije:n vuoden 2018 MLX90614 -kirjasto.
- Manuel Caballero:n vuoden 2017 MLX90614 -kirjasto.
- Sekä Rostislav Krylov:in vuoden 2021 MLX90614 -kirjasto.

Ratkaisu toteutui siten että käytimme Arnas MasRodije:n kirjastoa pohjana jota muokkasimme MLX90614 datalehden mukaisesti. Lopullinen kirjastotiedosto näyttikin seuraavalta.

```cpp
bool MLX90614::getTemp(float* temp_val){

    char outData[3];
    char inData[3];
    float temp_thermo;
    static int count = 1;

    i2c->stop();
    ThisThread::sleep_for(20ms);
    i2c->start();
    ThisThread::sleep_for(10ms);

    if (count == 1) {
        outData[0] = 0x00;
        i2c->write(i2c8bitaddress, outData, 1, true);
        ThisThread::sleep_for(10ms);
        count += 1;
    }

    // T obj 1
    outData[0] = 0x07;

    i2c->write(i2c8bitaddress, outData, 1, true);
    ThisThread::sleep_for(10ms);
    i2c->start();                           //repeat start
    i2c->write(i2c8bitaddress | 0x01);         //device address with read condition
    i2c->read((i2c8bitaddress | 0x01), inData, 3, true);
    ThisThread::sleep_for(10ms);
    i2c->stop();                            //stop condition

    temp_thermo=((((inData[1]&0x007f)<<8)+inData[0])*0.02)-0.01;      //degree centigrate conversion
    *temp_val=temp_thermo; // -273;                          //Convert kelvin to degree Celsius

    return true;                            //load data successfully, return true
}
```

Eräs ongelma todettiin myös MQTT viestinnän vastaanottamisessa mikro-ohjaimelta. Tämä ongelma tuli esille vasta JavaScriptin puolella, jossa yritettäessä kääntää payload dataa JSON JavaScript objektiksi, nousi error että ”Unexpected non-whitespace character after JSON data”. Tämä tarkoitti että JSON datan jälkeen mikro-ohjaimelta oli tullut jonkinlaisia extra kirjaimia, nämä puolestaan nostivat Errorin muuttamis-vaiheessa.



Kuvassa kyseessä ollut error sekä Payload data muutettuna string muotoon.

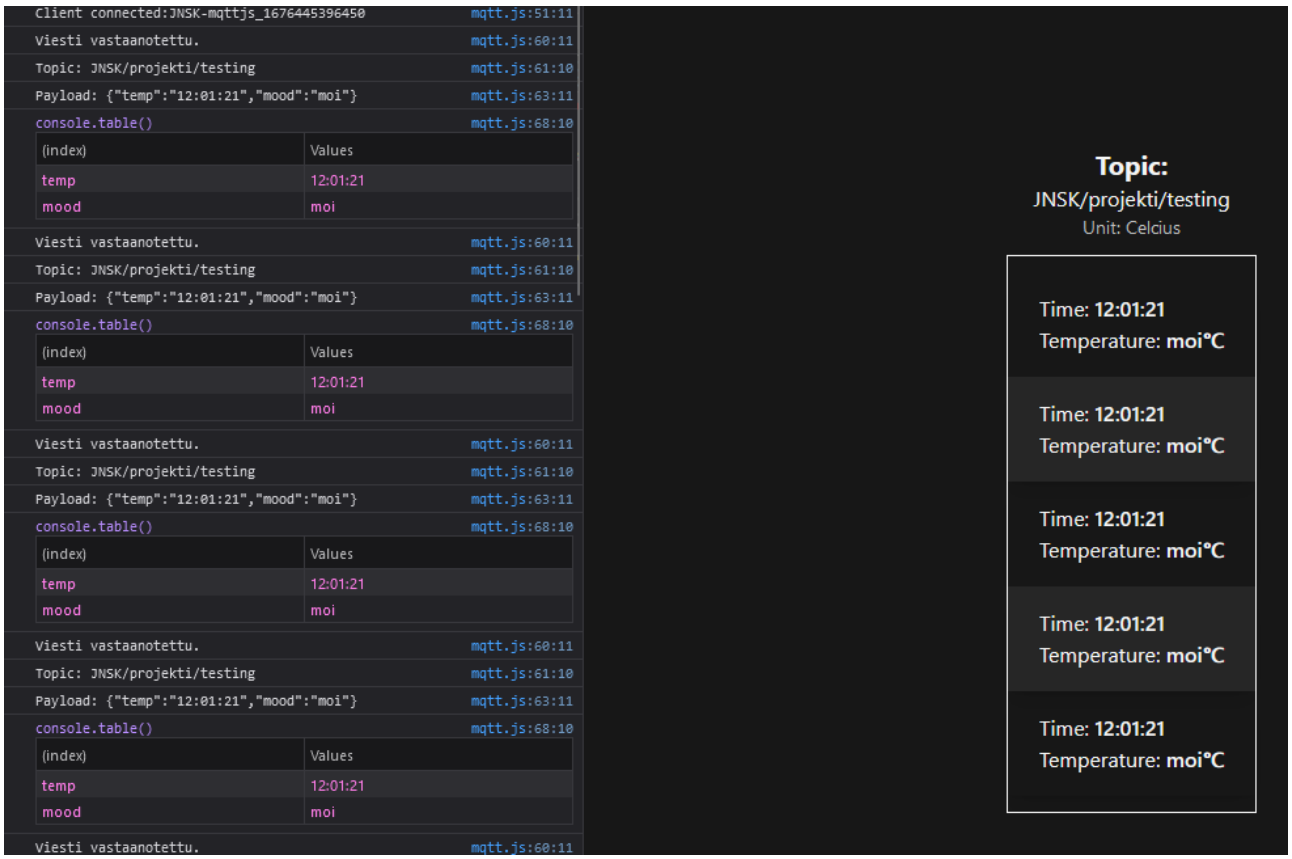
Ongelman syypää löytikin nopeasti sen jälkeen kun encodasimme payload datan luettavaan String muotoon JavaScriptin puolella. Tämä onnistui alla olevalla koodin pätkällä.

```
var payloadString = new TextDecoder().decode(payload);
console.log(payloadString) // "{"temp":"12:01:21","mood":"moi"}*************"
```

Tästä printistä huomattiinkin heti että mikro-ohjaimelta lähetetty payload sisälti monta turhaa merkkiä perässään. Seuraavaksi olikin siis muutettava mikro-ohjaimen koodia. Koodista löytyikin nopeasti ongelma, joka oli se että olimme muuttaneet payloadin pituudeksi staattisen 64 byten uint8array:n.

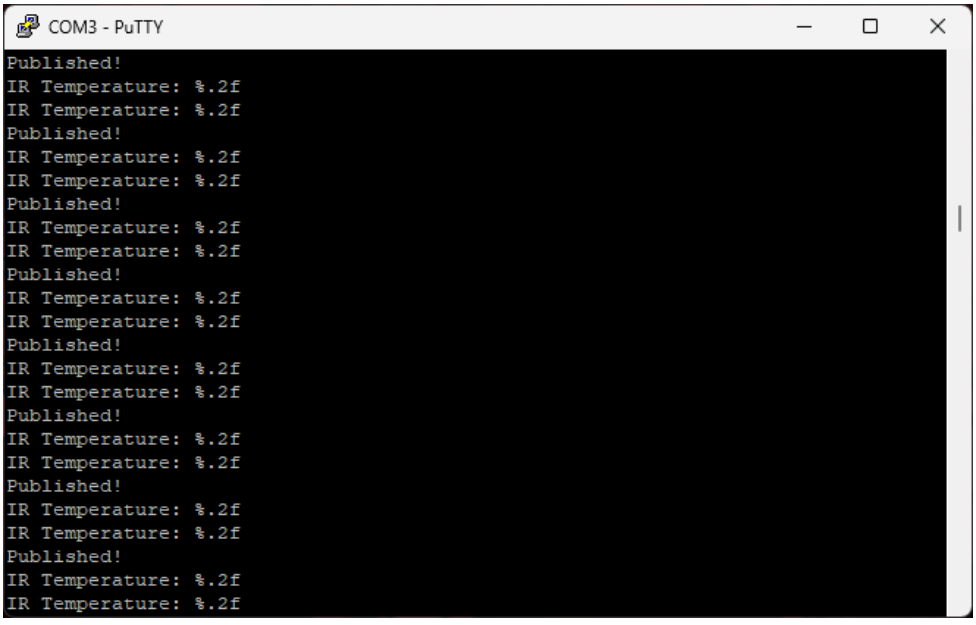
```
58
59      // msg.payloadlen = strlen(buffer);
60      // msg.payloadlen = 64;       <----- Alkuperäinen
61
62      socket.open(&esp);
63      socket.connect(MQTTBroker);
64      client.connect(data);
65
66
67
68      while(i <= 10) {
69          // sprintf(buffer, "Lasketaan kymmeneen! numero: %d", i);
70          sprintf(buffer, "{\"temp\":\"12:01:21\",\"mood\":\"moi\"}");
71          // update payload length to be the length of the message
72          // instead of being consistantly 64 bytes
73          msg.payloadlen = strlen(buffer);    // <------ Muutettu
74          client.publish(MBED_CONF_APP_MQTT_TOPIC, msg);
75          printf("Published! on topic\n\r");
```

Vaihdoimme payloadlen arvoksi staattisen ”64” sillä strlen(buffer) -funktio antoi meille error viestiä. Kuitenkin uudelleen yritettäessä, vaikka Keil Studio Cloud antoikin meille error varoitusta, se ei ollutkaan edes oikea error. Kun build:asimme ohjelman uudelleen ei nimittäin tullut yhtäkään ongelmaa ja ohjelma pyöri kuten halusimmekin. Laitamme tämän ongelman siis Keil Studion piikkiin. Emme ole vielä löytäneet syytä tähän ongelmaan. Mutta sen voin kyllä sanoa että tälläisten ongelmien myötä tulee vain ikävä Visual Studio Codea.



Vastaanotettu MQTT viestintä koodin muuttamisen jälkeen

Ongelma löytyi myös Floating Point numeroiden sisällyttämistä String dataan. Joten kun yritimme printata esimerkiksi konsoliin lämpötiloja float mudossa, niin saimme vain %.2f sen sijaan että olisimme saaneet oikeat arvot muuttujan kohdalle.



Kyseinen ongelma konsolin kautta luettuna.

Tähän löysimme korjauksen Mbed OS tuki forumilta. Ongelmaa olikin kysytty aikaisemmin otsikolla ”Float Printf() doesn’t work in desktop version”. Ongelma johtui siitä että Mbed OS 6.0:ssa oli luotu uusi asetus jonka työnä oli vähentää koodin tiedostokokoa. Tämä asetus oli nimeltään ”Minimal printf and snprintf”. Tämä asetus on kaikissa Mbed OS 6 versioissa valmiiksi päällä. Tämä ongelma ratkottiin lisäämällä yksi asetus mbed_app.json tiedostoon. Joka puolestaan poisti tämän ominaisuuden käytöstä.

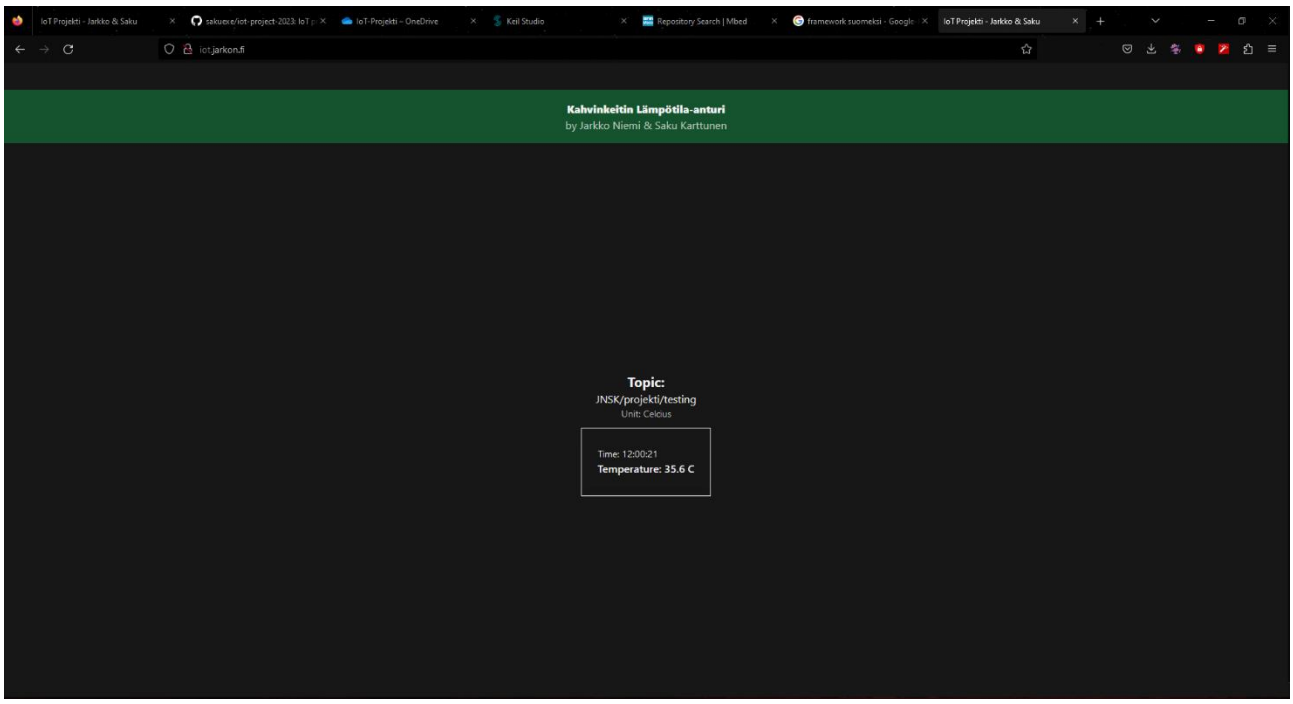
```
38          }
39      },
40      "target_overrides": {
41          "*": {
42              "platform.stdio-convert-newlines": true,
43              "esp8266.provide-default" : false,
44              "esp8266.built-in-dns": true,
45              "target.printf_lib": "std"
46          }
47      }
48  }
49
```

"target.printf_lib": "std" – rivin lisättyä floating point numeroiden printtaus onnistui.

4.1.2 Edistyminen, Frontend

Ongelmien hankaluuksia mukaanottamatta saimme silti paljon edistymistä projektin suhteen tehtyä. Käymme seuraavaksi läpi näitä edistyksiä.

Olimme jakaneet projektin toimintaa siten että Saku olisi vastuussa User Interface:n suunnittelusta ja toteuttamisesta. Ainakin suurimmalta osalta. Joten saimme rakennettua kehikon web-sovelluksellemme jota tulisimme käyttämään projektissa. Tämän kehikon työnä oli vastaanottaa MQTT viestejä ja muuttaa ne helposti luettavaksi käyttäjälle. Kehikko-versio rakennettiinkin siis vain HTML, CSS ja JavaScriptillä, mutta käytimme myös työkaluina TailwindCSS ja MQTT.js kirjastoja. Nämä auttoivat tekemään web sivun suunnittelusta ja rakentamisesta helppoa.



Kehikko-sivu ensimmäisellä viikolla.

Vastaanotetut MQTT viestit otetaan puolestaan vastaan, formatoidaan JavaScriptissä ja näytetään käyttäjälle näytön keskellä olevassa laatikossa. Tämä laatikko sisältää vain max viisi viimeisintä payload viestiä.

4.1.3 Edistyminen, Mikro-ohjain

Tämän jälkeen oli anturin asennuksen aika. Tämän me saimme aikaan maanantaina 15.2.2023 koulun IoT laboratoriossa. Ongelmat olemmekin ehtineet käydä läpi yksityiskohtaisesti joten seuraavaksi käymme lävitse mitä saimme aikaan mikro-ohjaimen puolella.

Saimme toteutettua ensimmäisen testaus sovelluksen anturin kanssa jolla näimme anturin lukemat lämpötilat tietokoneen näytöltä. Aluksi nämä arvot olivat hieman alhaisia, mutta saimme ne toimimaan paremmin kun vaihdoimme MLX90614 vielä kertaalleen.

Keskiviikkona 15.2.2023, mikro-ohjain oli Sakun mukana kotonaan, sillä meillä oli tullut vastaan ongelma MQTT viestien vastaanottamisessa UI:ssamme. Kokeilimme vielä kertaalleen uutta kirjastoa joka oli: Jens Strümper:in vuoden 2016 MLX90614 -kirjasto. Tämä kirjasto oli erilainen aikaisemmista, sillä se koostui aivan erinäköisestä koodista, joka oli suunniteltu helpommaksi luettavaksi. Yllätyksenä meille, tämä kirjasto toimi moitteitta heti ensimmäisellä yrityksellä.

Teimme siis hieman vielä muutoksia tähän kirjastoon, jotta saimme siitä vielä helpomman luettavan ja päädyimme käyttämään sitä.

Tässä vaiheessa meillä oli koossa siis mikro-ohjaimen anturin kirjasto ja luomamme demo-koodi toimi suurimmitta moitteitta. Joten seuraavaksi pitäisi vielä saada mikro-ohjain lähettämään tätä dataa eteenpäin MQTT viestinnällä. Mikro-ohjaimeen olimme kiinnittäneet jo aikaisemmin tunnilla WLAN kortin jolla pääsimme yhteyteen ulkoisen maailman kanssa. Käytimme siis Mbed Os:än MOD-WIFI-ESP8266 toimintaa.

Tämä toteutui helposti sillä lisäsimme vain aikaisemmalla tunnilla käytetyn Mbed-MQTT kirjaston projektiin, loimme mbed-app.json tiedoston ja toimme aikaisemmin käytetyn koodin uuteen projektiin. Tämä koodi yhdisti WLAN yhteyden ja alusti MQTT messagen meitä varten. Tässä vaiheessa prosessia ei koitunut muita ongelmia MQTT viestien lähettämisessä. Vain Keil Studio Cloudin kanssa. Sillä editori jostain syystä antoi error:eita koodimme vaikkei mikään ollut pielessä.

4.2 Viikko 2 (20.2.-24.2.2023)

Toisella viikolla keskityimme käyttöliittymän rakentamiseen sekä mikro-ohjaimen rakenteiden suunnitteluun. Tällä kertaa rakensimme käyttöliittymän ReactJS työkalulla jossa käytimme TypeScript ohjelmointikieltä.

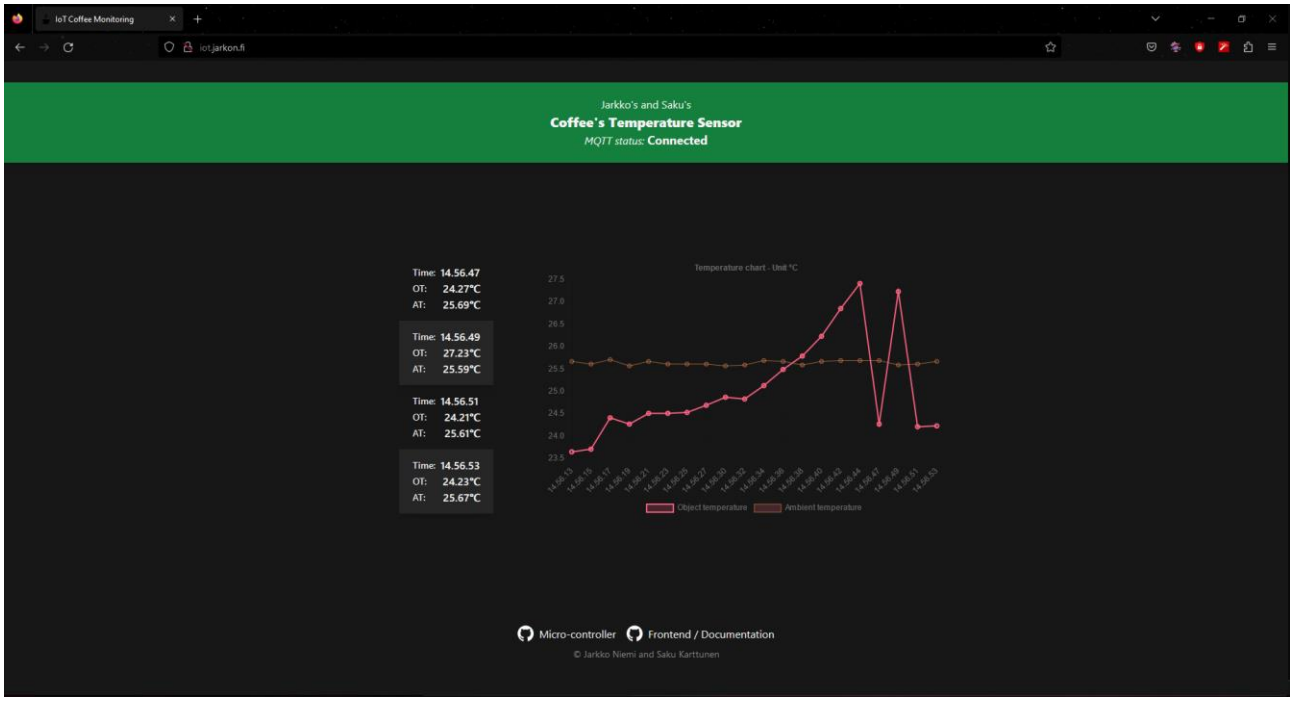
4.2.1 Ongelmat

Ongelmia tällä viikolla ei ollut monia. Yksi ongelma kuitenkin tuli vastaan kun Jarkko kloonasi mikro-ohjain koodimme GitHub repo:stamme. Koska meidän .gitignore tiedostomme esti MLX90614, sekä mbed-mqtt kirjastojen locaaleja tiedostoja pääsemästä Git:iin, niin kirjastot ladattiin uudestaan aina kloonaamisen yhteydessä. Tähän MLX90614 kirjastoon oli kuitenkin tehty aikaisemmin mainittu luettavuus muutos, niin rakennus vaiheessa read_temp() -funktio ei enään toiminutkaan. Sillä ladattu kirjastoversio piti sisällään alkuperäisen koodin. Tämä korjaantui kuitenkin sillä että Jarkko teki myös muutokset omaan versioonsa Keil Studiossa. Myös kirjasto-kansion poistaminen .gitignore tiedostosta olisi auttanut. Joten luettavuus korjauksemme ei ollutkaan täydellinen. Onneksi kuitenkin selvisimme säikähdyksellä, sillä mitään ei ollut rikki.

4.2.2 Edistyminen, Frontend

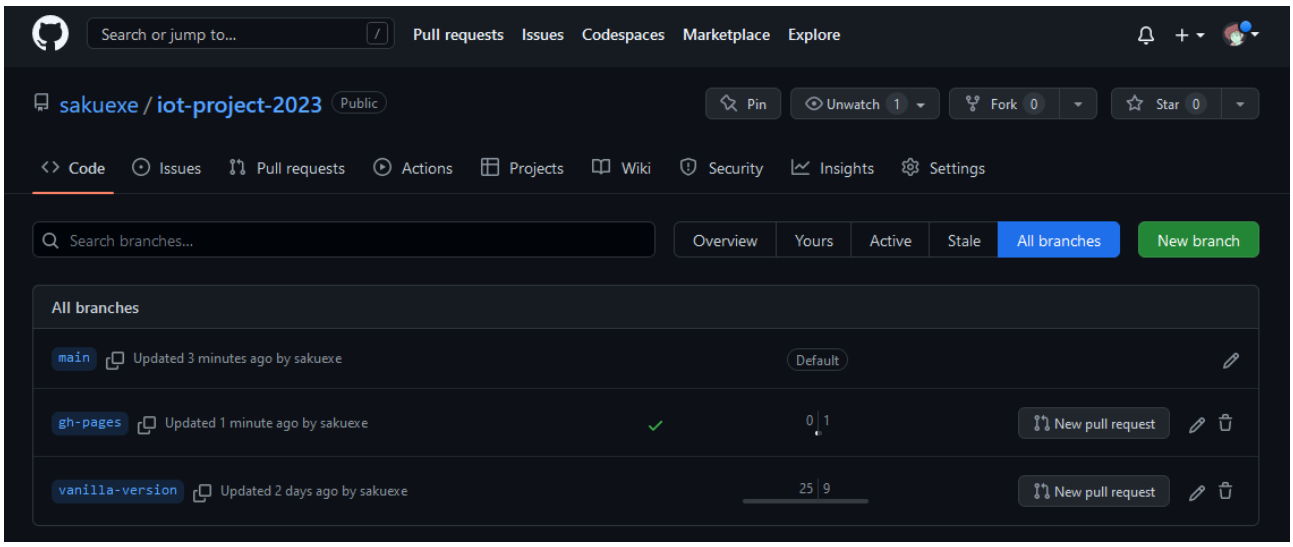
Toisella viikolla Sakun tehtävänä oli saada käyttöliittymä toimimaan kuten suunnittelimme. Vaihdoimme myös projektimme käyttämään React.js web-frameworkkiä. Tämän päätimme siitä syystä että saisimme hieman lisää harjoitusta React:in kanssa, sillä kesätyöpaikkoja etsiessä ollaan huomattu, että React:in osaaminen on tärkeä taito sovelluskehityksen alalla. React myös mahdollistaa paremmat työkalut yksisivuisten web-aplikaatioiden rakentamiseen, niin se sopi tähän tarkoitukseen hyvin. Tämän lisäksi käytimme myös aikaisemmin mainittua TailwindCSS framework:iä nopeampaan CSS:än kirjoittamiseen. Reactissa käytimme vielä TypeScript:iä kielenä käyttämällä seuraavaa komentoa projektin rakentamis vaiheessa:

```
npx create-react-app todo-app --template typescript
```



Uusi käyttöliittymä React:illa rakennetussa web sovelluksessamme.

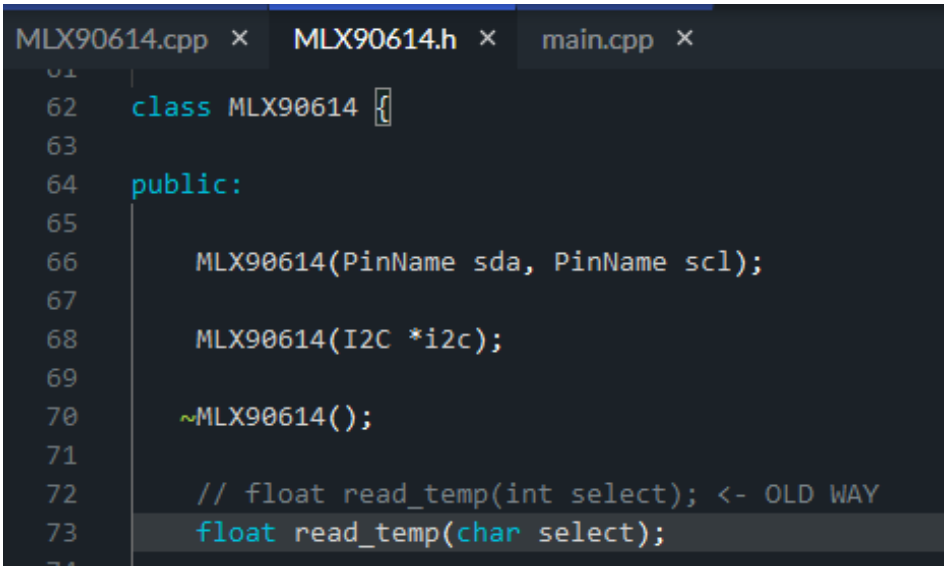
Teimme myös frontend Github repo:on muutoksia. Lisäsimme kaksi branchia repoon nimillä ”vanilla version” ja ”gh-pages”. Vanilla versionissa olisi aiemmin luotu kehikko sivu ja gh-pages on puolestaan paikka jossa pidämme ”production” koodiamme. Tämän koodin saamme käyttämällä komentoa ”npm run build” tämän jälkeen committaamme build kansion sisällöt ja pushaamme sen gh-pages branchille. Github pages valikosta valitsemme vain oikean branchin ja tämän jälkeen sivumme ovatkin julkisia.



Kuvassa github repomme eri branchit.

4.2.3 Edistyminen mikro-ohjain

Muutimme mikro-ohjaimen käyttämää lämpötila anturi koodia hieman, jotta sitä olisi helpompi lukea ja täten käyttää koodissamme. Muutimme sitä siten, että kun read_temp() -funktiota käytetään, parametriksi annetaan joko 'A' taikka 'O'. Tämän avulla voi helposti hakea "Ambient" sekä "Object" lämpötilat ja koodikin pysyy helpommin ymmärrettävänä. Alkuperäiset arvot olivat 0 sekä 1. Tämä oli ongelmallista aina kun muutimme koodia, sillä se aina tahtoi unohtua. Mietimme myös asiaa kolmannen tekijän suunnasta ja päätimme tehdä korjauksen jotta myös muilla olisi helppo ymmärtää ja soveltaa mitä koodilla tehdään.



MLX90614.cpp × MLX90614.h × main.cpp ×

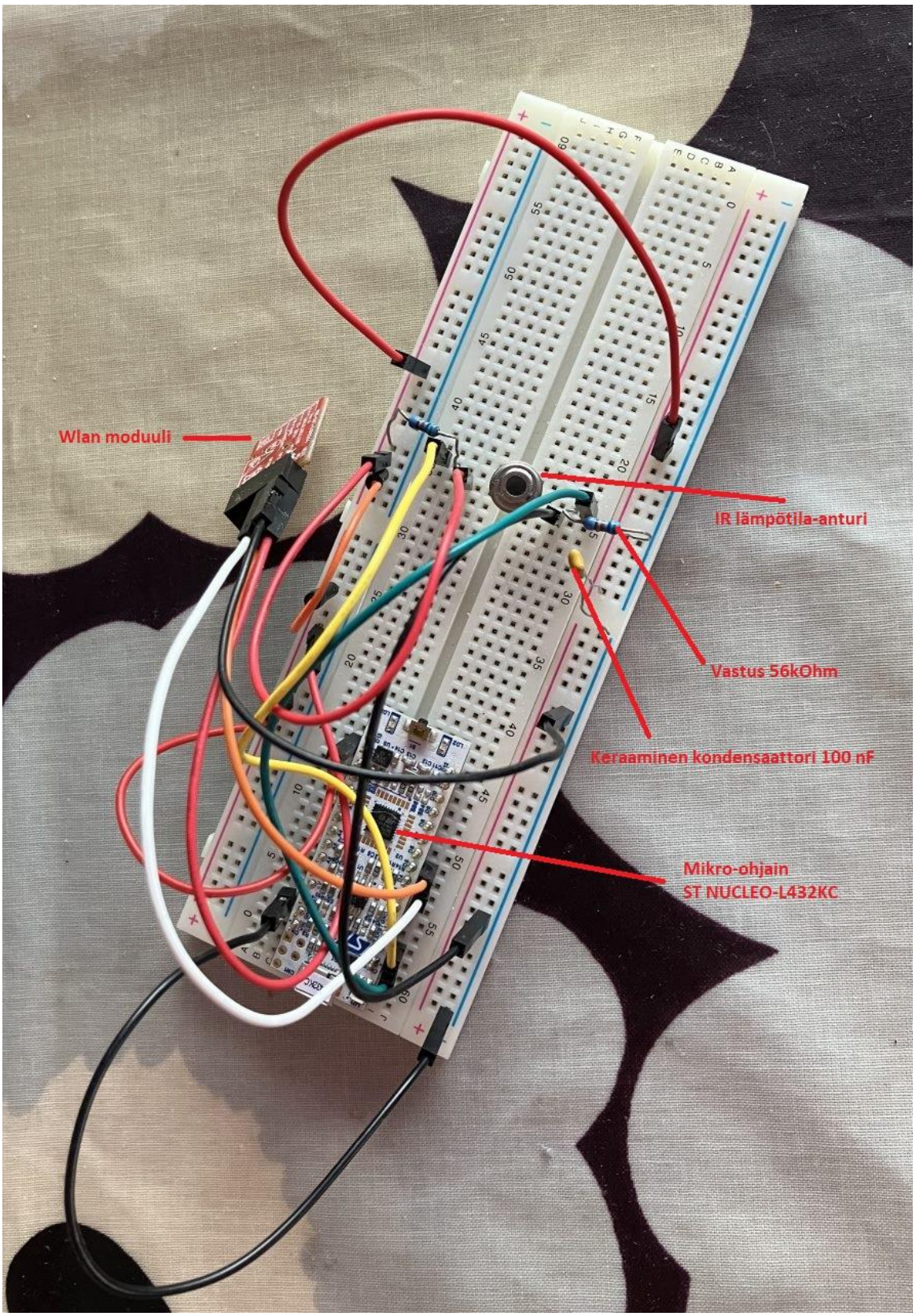
```
22      return 0.02 * static_cast<float>((cmd[1]<<8)|cmd[0]);
23  }
24
25  float MLX90614::read_temp(char select) {
26      uint8_t reg_addrs[] = { T_ambient, T_obj1 };
27      float tt = 0.0;
28      if (select == 'A'){
29          tt = get_temp(reg_addrs[0])-273.15;
30          }
31      if (select == 'O'){
32          tt = get_temp(reg_addrs[1])-273.15;
33          }
34      return tt;
35      }
36
```

Tekemämme muutokset anturin kirjastoon

MLX90614.cpp × MLX90614.h × main.cpp ×

```
99      printf("START\r\n");
100
101     for (int i = 0; i < 20; i++) {
102
103         // if green LED1 is on, that means temperature is being
104         // read from the sensor
105         myled = true;
106
107         // Read object temp first
108         objectTemp = thermometer.read_temp('O');
109
110         // for making sure that the floating point printing works,
111         // the following line was added to the mbed_app.json
112         // "target.printf_lib": "std"
113         // this disables Mbed OS 6.0's 'minimal printf and snprintf'
114         printf("IR Temperature: %.2f \r\n", objectTemp);
115         ThisThread::sleep_for(20ms);
116
117         // read ambient temperature second
118         ambientTemp = thermometer.read_temp('A');
```

Käyttämämme koodi main.cpp tiedostossa muutoksien jälkeen.



Kuva leipälaudasta jossa kytkettynä tarvittavat komponentit.

Mikro-ohjaimen testaamista oikeassa käyttötilanteessa.

Olemme suorittaneet mikro-ohjaimen testauksen käyttäen keitettyä vettä, jotta voisimme saada tuloksia todellisesta käyttötilanteesta. Anturi toimitti järkeviä arvoja varsinkin silloin, kun se oli riittävän lähellä kahvipannua. Tämä johtuu siitä, että lämpötila-anturin keila on laaja (90 astetta).

Laitteen ulkoasua on mahdollista vielä parantaa. Tarpeettoman pitkät johdot tulisi lyhentää oikean pituisiksi ja komponenttien mahtuminen pienempään piirilevyyn on toteutettavissa. Tarkoituksena on tehdä nämä muutokset projektiviikolla 4.

4.3 Viikko 3 (27.2. – 3.3.2023)

Viikko 3 oli HAMK:in hiihtoloma viikko, joten emme työskennelleet projektin parissa.

4.4 Viikko 4 (6.3.-9.3.2023)

Viimeinen viikko IoT projektia piti sisällään viimeisten ongelmien hiomista sekä raportin viimeistelyä. Torstaille suunnittelimme myös esitystä. Tämän takia viimeisellä viikolla ei ollut paljoa uutta mitä olisimme rakentaneet taikka kehittäneet projektimme parissa. Suurin kehitys tapahtui mikro-ohjaimen rakentamisessa, kun siirsimme sen leipälaudalta oikeaan pakettiin. Verkkosivujen suhteen ei isoja muutoksia

4.4.1 Ongelmat

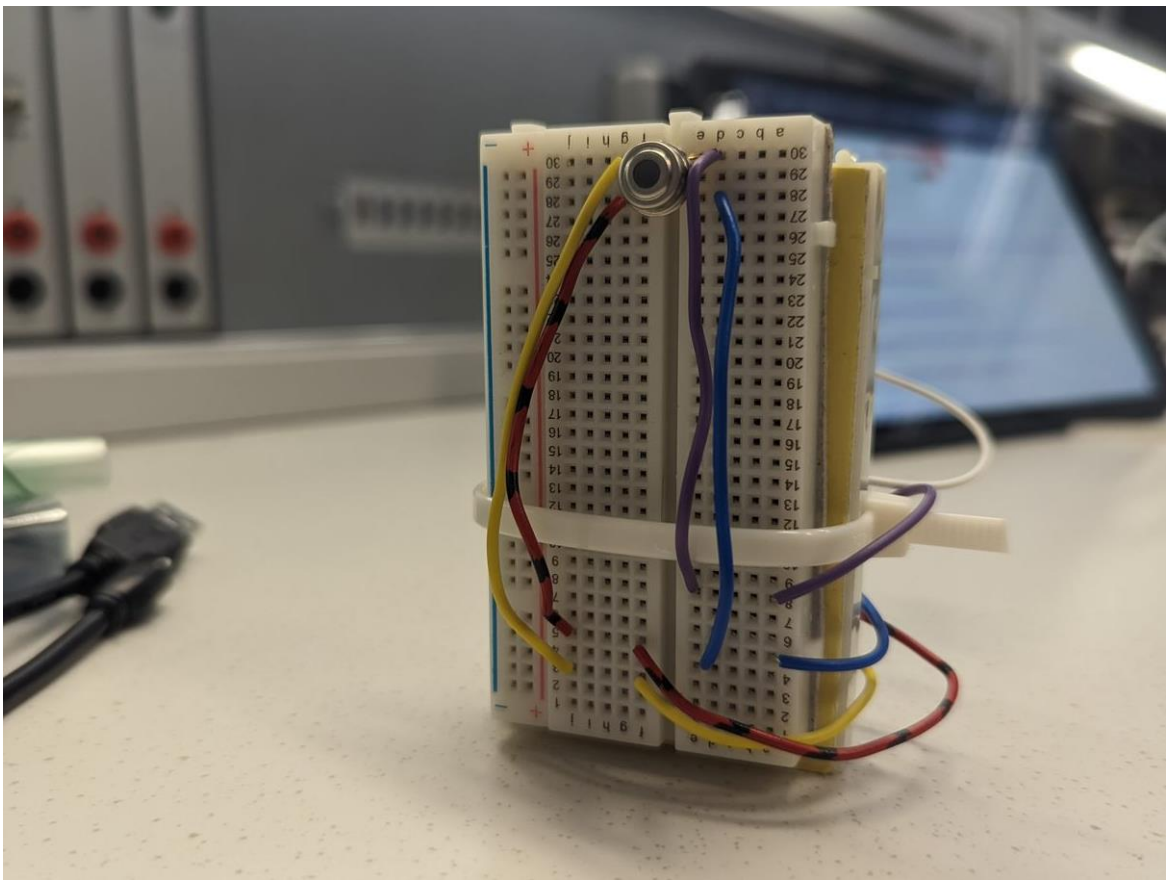
Viikolla 4 ei tullut vastaan ongelmia ja pärjäsimme tehtäviemme kanssa hyvin. Jos jotain pitäisi keskiä tosin, niin mikro-ohjaimen ulkonäön ja muotoilun muuttaminen oli paikoittain haastavaa. Tämä johtui suurimmalta osalta siitä, että m-f liitäntäpiuhojen liitännät eivät tarttuneet IR anturin kovinkaan tiukasti, joten emme voineet yhdistää anturia pelkästään piuhojen päähän.

4.4.2 Edistyminen

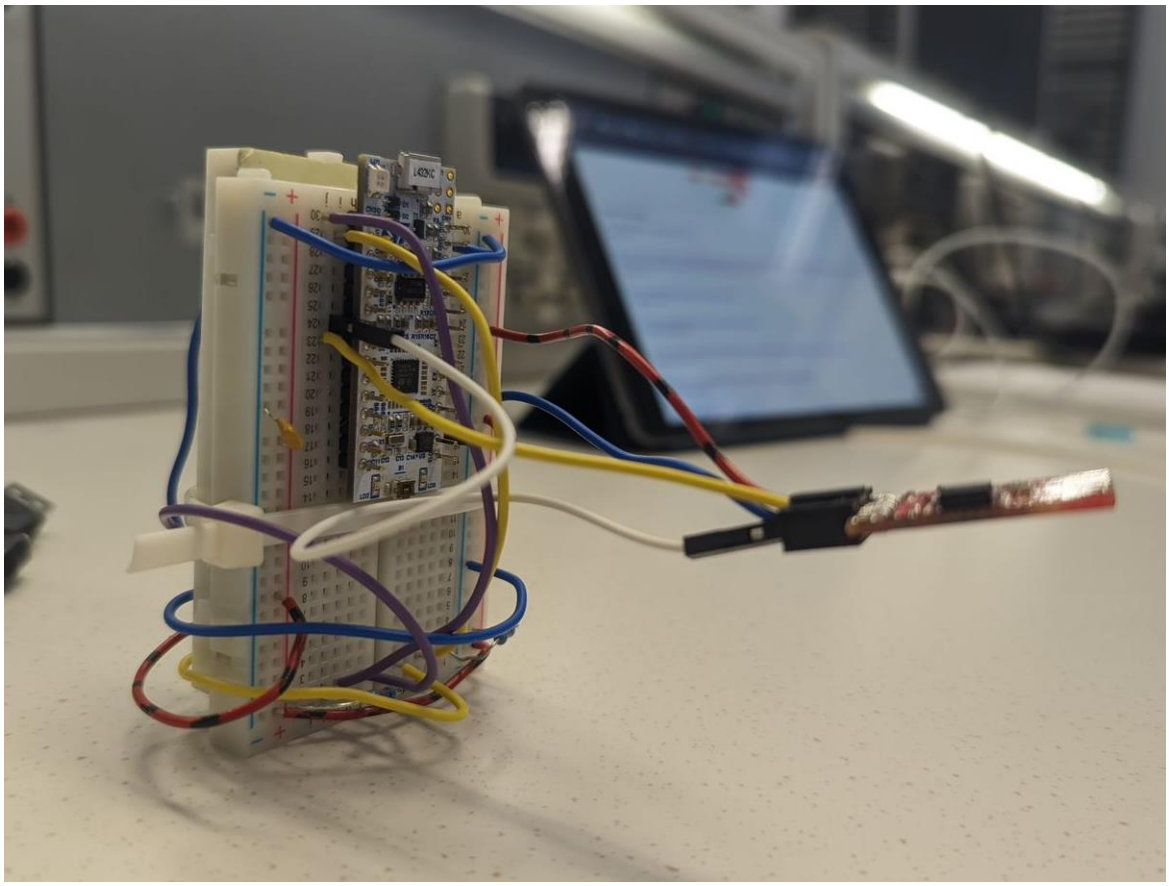
Projektin loppusuora edistyi hyvin. Viimeistelimme tämän raportin ja korjasimme viimeisiä bugeja käyttöliittymästä sekä mikro-ohjaimesta. Loimme myös mikro-ohjaimelle oikean ”paketin” jotta saimme sen paremmalle alustalle käyttöä varten.

Lopullinen muotoilu oli lyhyempi, kompaktimpi sekä siinä oli vähemmän pitkiä piuhoja jotka tulisivat eteen käytössä. Uuden muotoilun piuhat ovat eri väriset kuin prototyypissä, joten piuhoja kannattaa verrata aiempaan versioon jos aikoo rakentaa itse projektimme. Lisäsimme toisen leipälaudan alkuperäisen taakse, jotta infrapuna anturi olisi helpompi suunnata oikeaan kohteeseen. Tämä mahdollisti myös puhtaamman signaalin infrapunalle kuljettavaksi.

Mikro-ohjaimen käytöstä saimme täten helpomman ja mutkattomamman.



Mikro-ohjaimen lopullinen muotoilu edestä (IR Anturin puoli)



Mikro-ohjaimen lopullinen muotoilu takapuoli (ohjainpuoli)

5 Ratkaisun analyysi

Toteutuksemme toimi hyvin suunnitelman mukaisesti, vaikkakin matkalta löytyi ongelmia jotka yrittivät vaikeuttaa työtämme. Saimme mikro-ohjaimen mittaamaan anturin eteen laitettua kohdetta kuten pitääkin sekä huoneen lämpötilankin saimme anturin laskemaan. Nämä anturin mittaavat ja laskevat arvot saimme lähetettyä MQTT protokollalla helposti itse luodulle verkkosivulle jossa dataa oli helppo muuttaa visuaaliseksi käyttäjää varten.

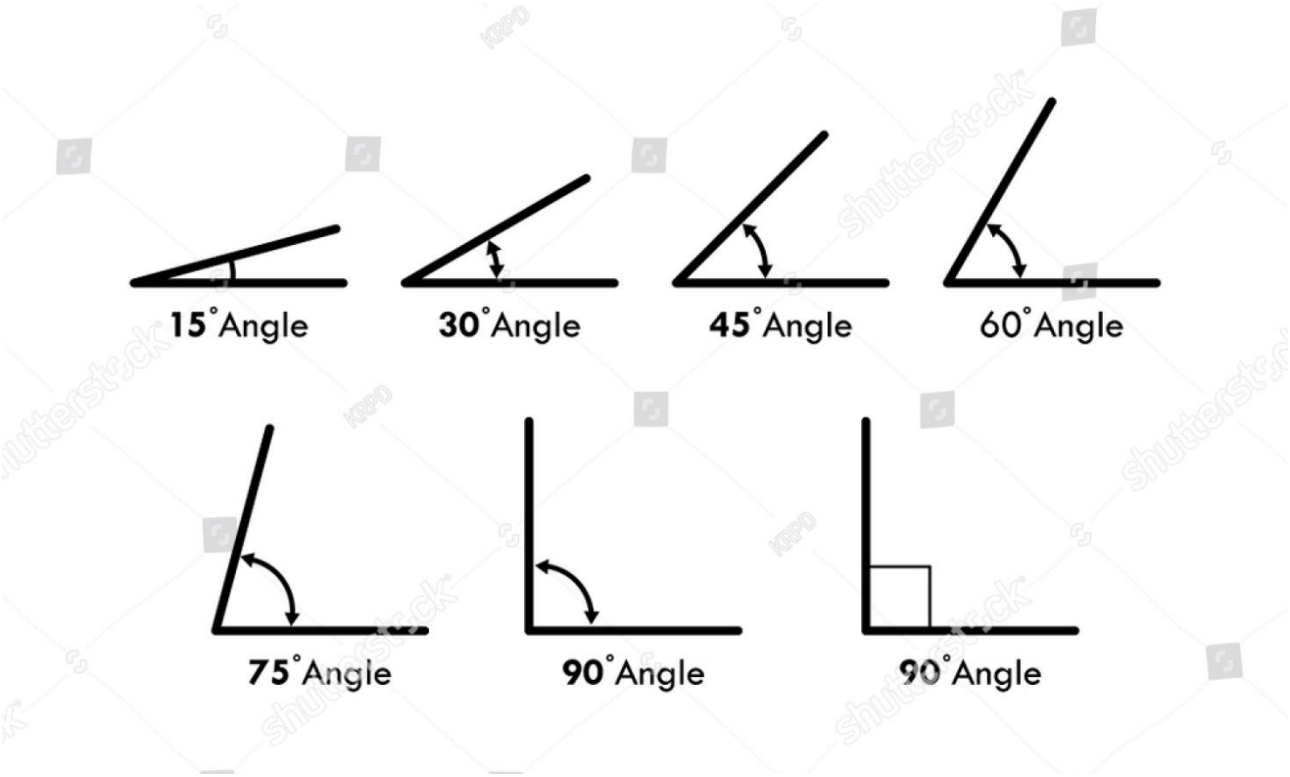
Jos meillä olisi aikaa ollut enemmän, niin olisimme miettineet tuotteen tuotantopuolta enemmän. Tällä tarkoitamme, että Web-sovellukseen olisi voinut luoda oman valikkonsa, johon asiakas olisi saanut omat tunnuksensa. Nämä tunnukset olisivat sitten auttaneet valitsemaan käyttäjän oman mikro-ohjaimen viestinnän. Kuitenkin projektissa kappaleita oli vain yksi, niin emme tuotantoa suunnitelleet pidemmälle.

5.1 Kritiikit ja muutokset

Mietimme myös tässä otsikossa että jos tekisimme projektimme uudelleen, mitä asioita muuttaisimme tai tekisimme eri tavalla.

Ensimmäinen asia jonka keksimme oli että hankkisimme saman valmistajan anturin, mutta jossa olisi pienempi FOV (Field of View). Tämä tarkoittaisi sitä että anturin lähettämä ”Keila” olisi tiukempi. Tämä puolestaan mahdollistaisi sen, että mikro-ohjain olisi helpompi pitää hieman kauempana mitattavasta kohteesta. Nimittäin projektissa käytetyssä MLX90614ESF -infrapuna lämpöanturissa on FOV 90°. Tämä pakottaa mikro-ohjaimen pitämistä maksimissaan noin 30cm päästä kahvinkeittimestä, jotta mittaustulokset ovat käyttötarkoitukseen tarpeeksi todellisuutta mukailevat.

Joten jos tekisimme projektin uudestaan, haluaisimme käyttää noin 30° taikka jopa 10° versiota. Tämä mahdollistaisi huomattavasti pidemmän toimintamatkan mikro-ohjaimellemme.



Esimerkkikuva FOV -kulman visuaalisointia varten.

Samalla miettisin seuraavalla kerralla että onko ReactJS tarpeellinen työkalu tähän projektiin ja että tuoko se liikaa ”turhaa täytettä” käyttöliittymään. Omasta mielestäni käyttöliittymä ei kärsinyt ReactJS:än käytöstä, se on silti nopea ja responsiivinen. ReactJS:än käyttö olikin suurimmalta osalta siitä syystä että halusimme lisää kokemusta sen käytöstä. Ei siis siitä syystä että projekti pakottaisi ReactJS:än käyttöä. Olemme kuitenkin tyytyväisiä että käytimme sitä, sillä projektin aikana ehti oppia paljon sen käytöstä. Varsinkin TypeScript:iä käyttäessä.

6 Yhteenveto

Yhteenvetona halusimme siis tehdä mikro-ohjaimen jonka avulla voisimme seurata kahvikoneen kahvin tippumista jotta saisimme tietää optimaalisen ajan kahvin nauttimiselle. Tämän me toteutimme kehittämällä mikro-ohjaimen jossa käytimme MLX90614 Infrapuna -lämpötilaanturia. Halusimme käyttää infrapuna-anturia lämpötilan mittaamiseen, sillä sen avulla voisimme tehdä mittaamisen etänä mittauskohteesta. Toinen mahdollinen anturi olisi ollut ”Digilent Pmod TC1: K-type thermocouple” (Digilent). Isoimmat erot näiden kahden antureiden välillä olivat se että infrapuna anturia ei tarvitsisi pestä jokaisen käytön jälkeen, sillä sitä ei upotettaisi kahviin, eikä sen tarvitsisi olla kontaktissa mittauskohteen kanssa. Infrapuna-anturi mahdollistaisi siis helpomman käytön loppukäyttäjälle, sillä sen tarvitsee vain liittää voima lähteeseen (3.3V) USB-MicroUSB

piuhalla ja se alkaa toimimaan. Tämä mahdollisti siis käyttäjälle helpon ”Plug-and-play” käyttötuntuman.

Käyttäjäkokemus oli projektissa aina isona arvona alusta saakka. Suunnitelmasta halusimme toteuttaa tuotteen siten että hypoteettinen asiakas jolla ei ole aikaisempaa osaamista tekniikasta, voisi ostaa tuotteen ja saada sen toimimaan helposti. Tämän arvomaailman huomaakin suunnitelmasta toteutukseen saakka. Käyttäjän mukavuus oli myös otettu mukaan käyttöliittymässä. Aluksi suunnittelimme että tekisimme sovelluksen jonka käyttäjä lataa, mutta huomasimme heti että tässä olisi ongelma kehityksessä kun pitäisi tehdä omat Android ja IOS sovellukset. Päädyimme siis luomaan käyttöliittymän Web App:inä. Tämä mahdollistaisi sen että sillä ei olisi väliä että onko käyttäjä puhelimella, koneella taikka tabletilla, niin käyttäminen onnistuisi helposti ilman mitään lataamisia.

Onnistuimme projektissa omasta mielestämme hyvin ja saimme toteutettua suunnitelman juuri niin kuin toivoimmekin. Mikro-ohjain toimi ja oli helppo saada mittaamaan lämpötilaa ja käyttöliittymä oli selkeä sekä helppo käyttää.

Tunnemme että oman Web-sovelluksen luominen ja kehittäminen mahdollisti paljon paremman käyttökokemuksen kuin jos olisimme käyttäneet NodeRed:iä. Sillä Web-sovelluksen käyttöön ei tarvitse ladata mitään, joten kuka vain voi avata sivun ja seurata kahvin lämpötilaa omalta valitulta laitteeltaan. Tämä oli tärkeä saada toteutettua sillä kuten aikaisemmin kävimmekin lävitse, niin käyttömukavuus ja helppous oli todella iso prioriteettimme projektin alusta alkaen. Joten voimme itsevarmasti sanoa että onnistuimme tämän toteuttamisessa.

Liitteet

Liite 1 – Mikro-ohjaimen tekniset piirrustukset