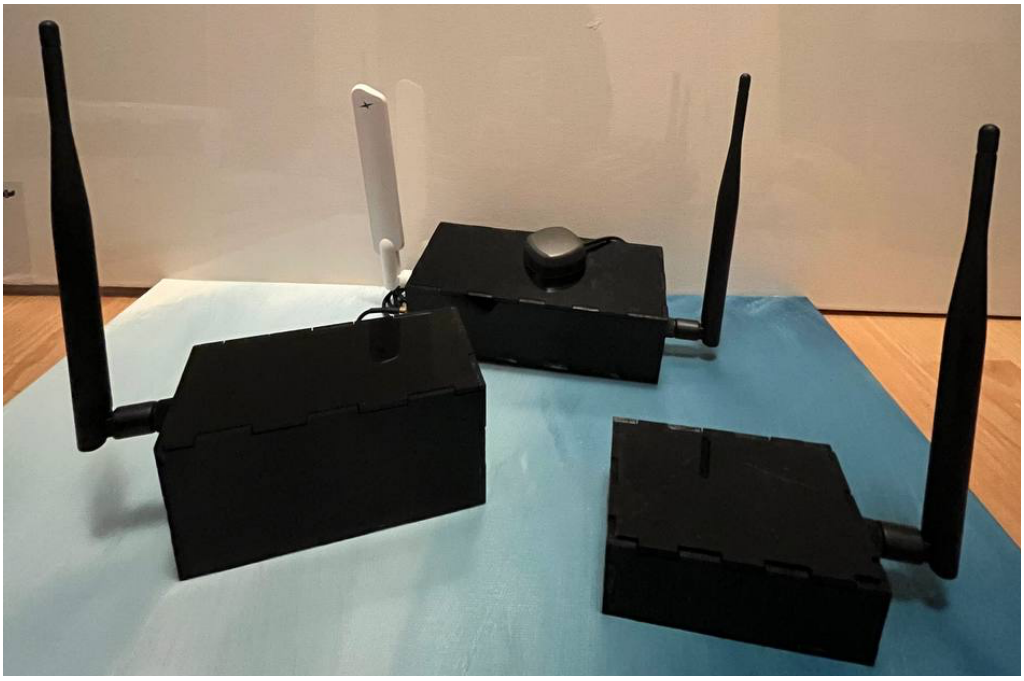
Aalto-yliopisto, Sähkötekniikan korkeakoulu
ELEC-D0301 Protopaja
2023

Final report

Project #02 (Granlund)
Network Base Station Monitor



Date: 26.07.2023

Beatriz Glaser
Mimi Määttä
Akbar Urlenbayev

Information page

Students
Akbar Urlenbayev
Beatriz Glaser
Thuy “Mimi” Määttä

Project manager
Beatriz Glaser

Sponsoring Company
Granlund Oy

Starting date
01.06.2023

Submitted date
07.09.2023

Abstract

This report outlines the development and implementation of an Internet of Things (IoT) device designed to monitor structural parameters of network base stations, often located in remote and hard-to-reach areas. The device was developed during Aalto University's Protopaja course with the aim is to digitize and automate the monitoring and analysis of various structures, products, and services. The primary objective was to create an energy-efficient, long-lasting IoT device capable of measuring the structural conditions of the station towers and display the collected, clean data to the user.

Utilizing wireless sensor stations, the device captures real-time data and transmits it to a microprocessor via Bluetooth. Data is then sent to a cloud-based server via a mobile network, where it is processed, stored, and displayed through a user interface. This allows for real-time statistical display, anomaly detection, and predictive maintenance alerts.

The block diagram of the system is shown below:

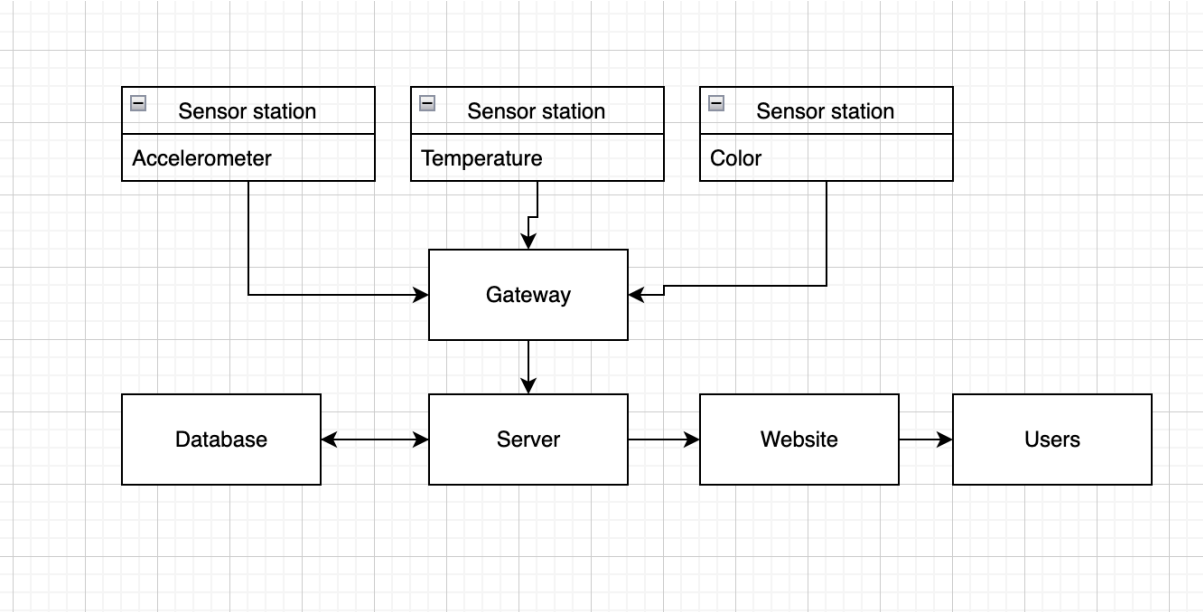


Table of Contents

1. Introduction / Johdanto

Network base stations are often located in remote areas which are hard to reach. The maintenance and replacement of components of these towers tend to be expensive and resource consuming. Our goal is to create an IoT device that will monitor structural changes to the tower and send real-time data to a server, which will display this information to the user. Having this kind of information means that extensive analysis can be done to predict needs for maintenance or replacements, as well as gather other statistical data. By monitoring the towers remotely, it is possible to save costs and other resources. Moreover, we will be able to make better predictions about tower placement and understand the effect of external conditions (e.g., wind and cold) on the structure.

This automation and digitization of base station maintenance procedures is part of a bigger project called GenerIoT (https://itea4.org/project/generiot.html), where companies in different countries are applying the same goal of facilitating monitoring and analysis of structures, products and services in different areas.

2. Objective / Tavoite

The main goal of this project is to create an IoT device that uses sensors to gather data from the structure of a network base station (those measurements being the deflection of the tower, the material temperature and the tower's signal light), and securely sends this information to a server which processes the data and displays the concluded information. Additionally, we aim to provide predictions on maintenance and alert users of data anomalies. Lastly, we created an easily scalable product with the possibility of adding more sensors and measurements.

Our target users are mobile network companies, who would use it mainly for maintenance of base station towers. Moreover, with some modifications, this kind of project can also be applied to other structures (such as wind turbines).

The main functionality of our device is collecting accurate data and providing remote access to it in real time. Additionally, given its usage specifications, the device should be power efficient and have a long-lasting lifetime. We would also like to provide a consistent and smooth user experience.

In technical terms, our goal with this project is:

- Gather data from light function, tower temperature and deflection (using wireless sensor stations)
- Transmit the measurements from the sensors to a microprocessor (on our gateway) using bluetooth connection
- Use energy harvesting technology to power the sensor stations and gateway
- Have a safe enclosure for the gateway device
- Transmit data from the gateway to a cloud server using mobile network
- Receive, process and store the data using a database
- Provide a user interface for statistical display, anomaly notifications and calculated predictions.

3. Hardware

3.1. Gateway

3.1.1. Components

The gateway gathers the data that is collected from sensors, then sends the data to our server for further analysis. For these purposes, we use 2 different modules. The measured data collected from the sensor is sent to the gateway using a bluetooth module, the nRF52832 system-on-chip (SoC). Due to the advantage of the BMD-301, which is based on the nRF52832 SoC by Nordic Semiconductor, it is used in our gateway. The BMD-301 features an ARM® Cortex™ M4F CPU, an integrated 2.4GHz transceiver, and is also equipped with a U.FL connector to support an external antenna. Figure 1 shows the block diagram of the BMD-301 built on the nRF52832. In addition, we also need a bluetooth antenna for this module. Based on the recommended antennas for this module, we selected an antenna with max gain of 5.0 dBi and 1/2 wave type. The antenna also has an ingress protection IP65 which gives the antenna dust and water resistance.

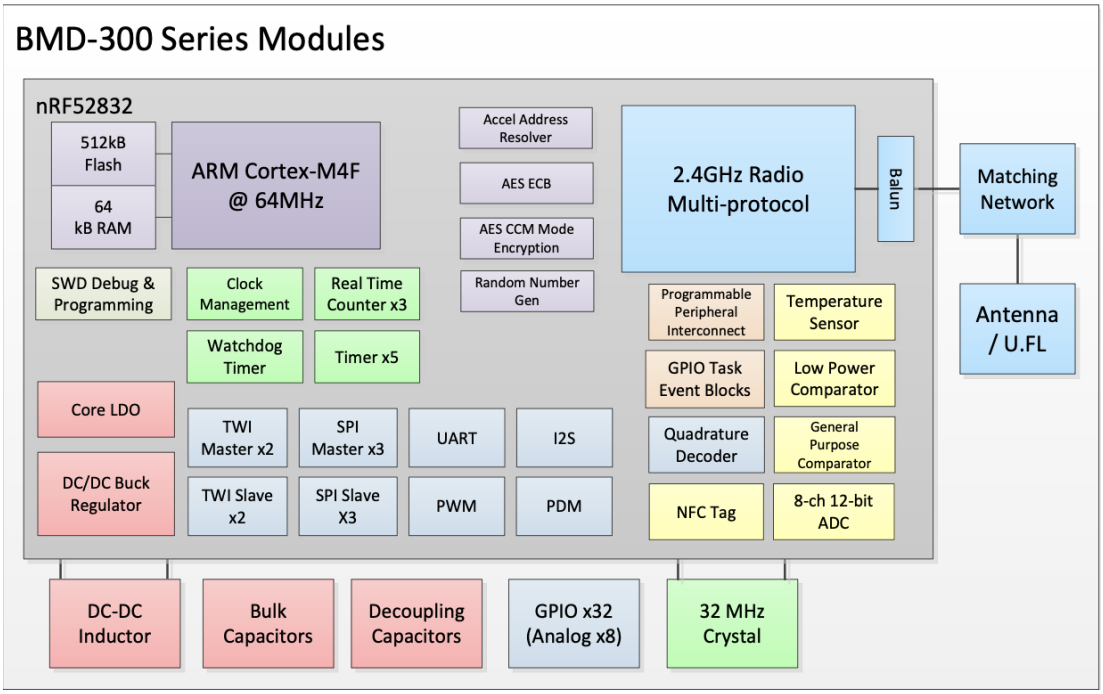


Figure 1: Block diagram of the BMD-301 (Src: BMD-301 data sheet)

For sending data from the gateway to the server, we use LTE-M (Long-Term Evolution for Machines) cellular module, more specifically, the Icarus IoT v2 Board is used (which was built around nRF9160 SoC by Nordic Semiconductor). The board provides low power consumption and is well-suited for IoT devices that need to transmit small amounts of data over long distances. Figure 2 shows the block diagram of the Icarus IoT v2 board. Additionally, LTE antenna and GPS antenna were selected based on the board's recommendation with specific frequency and frequency range. Both antennas have SMA connectors for easy access and maintenance.

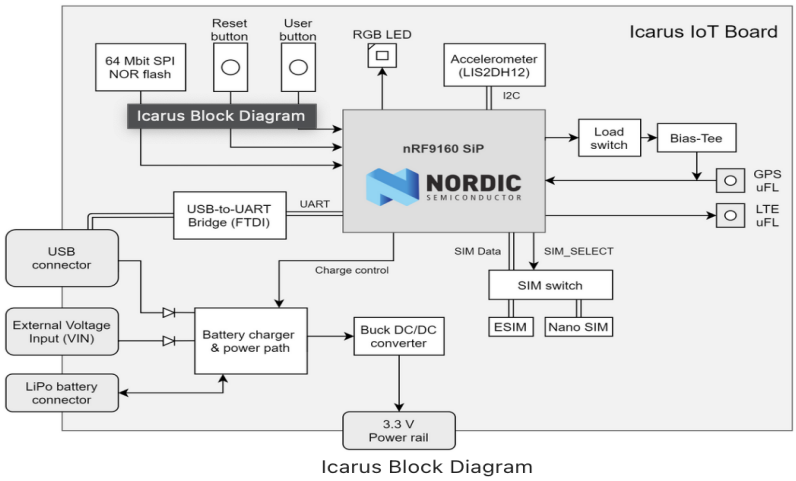


Figure 2: Block diagram of Icarus IoT board (Src: Icarus Data sheet)

3.1.2. Circuit and PCB design

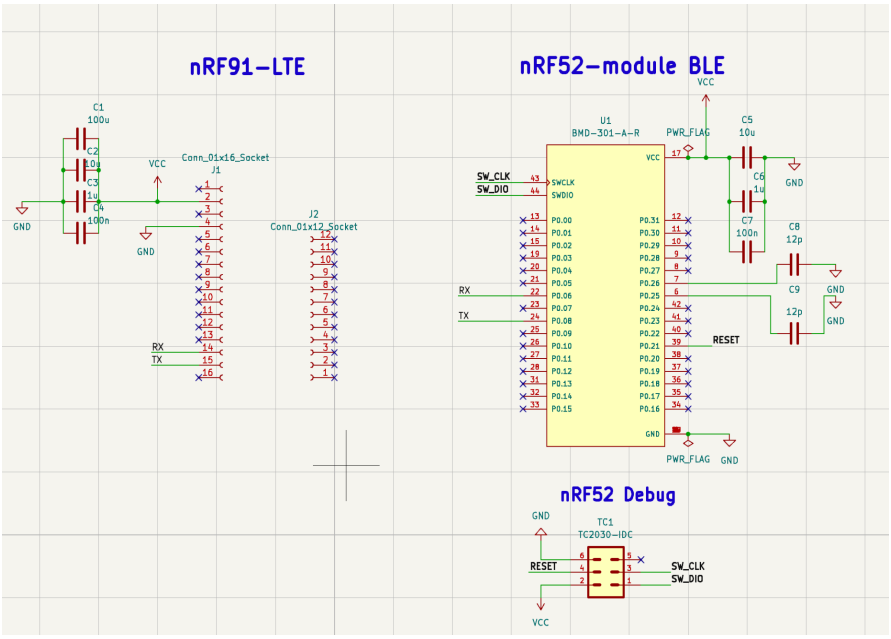


Figure 3: Circuit diagram of the gateway

The circuit and PCB were designed based on the modules' requirements and guidelines. A tag connector was added to the PCB for programming and debugging the nRF52 chip. The bluetooth and the LTE modules communicate to each other using UART protocol, which was implemented on the schematic and PCB.

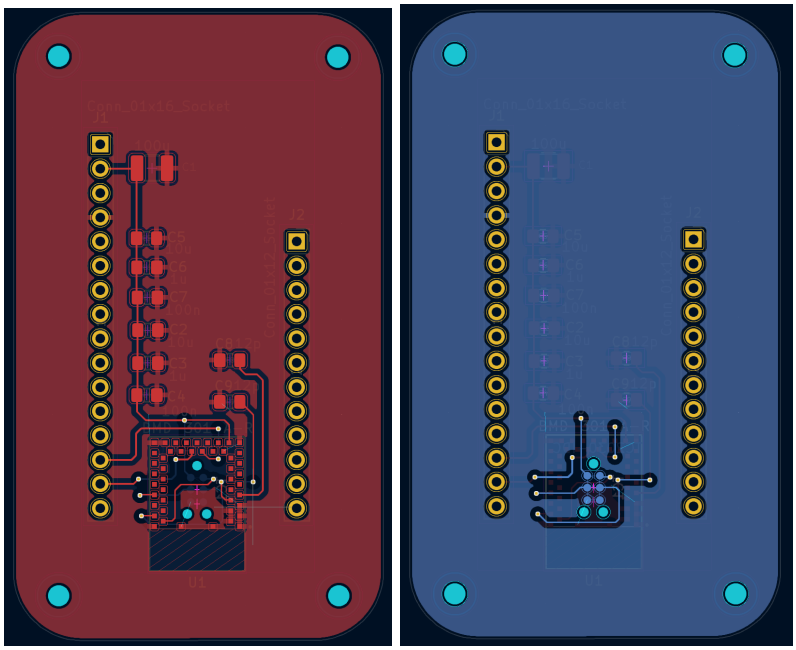


Figure 4: PCB of the gateway with layer 1 (left) and layer 2 (right)

However, after seeing a potential issue with the above design, it became evident that the initial placement of the programming tag could give rise to accessibility challenges. Therefore, a decision was made to relocate it to a more distant location, and ensure that there would be no issue to connect the programming tag. The new design is shown in the image below.

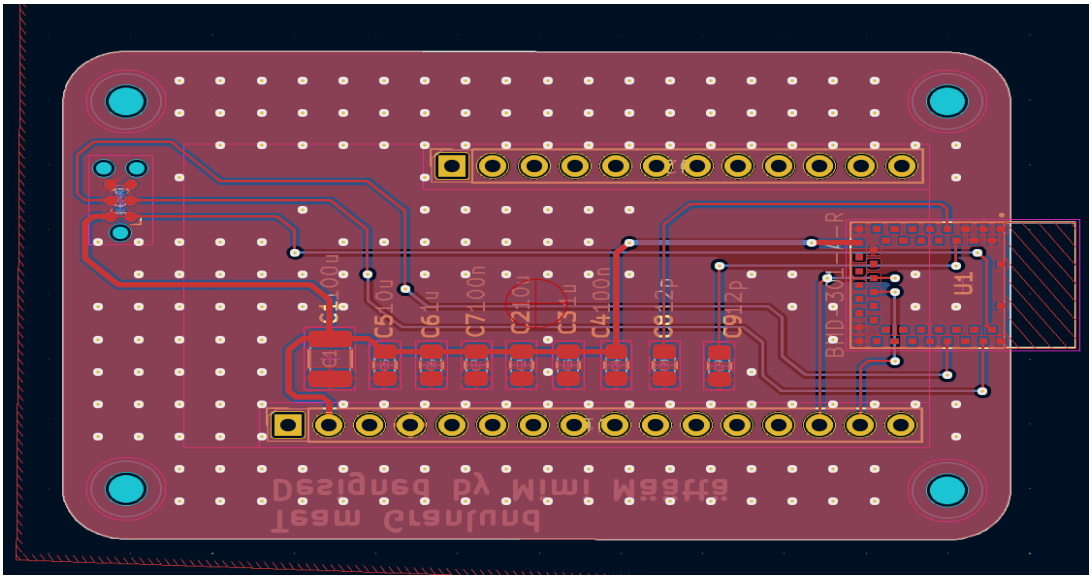


Figure 5: PCB of the gateway - redesigned

3.2 Sensors

For the goal of this project, we decided to make 3 different wireless sensor stations, which will detect the activity of the tower signal light, material temperature, and the movement of the cell tower. Each sensor station will be placed in a different spot on the tower. The signal light sensor will be placed facing the signal light near the top of the tower. The accelerometer sensor station should also be placed on top of the tower for the greatest deflection values. Finally, the temperature sensor station does not have such strict requirements and can be placed anywhere on the network base station as long as it is in contact with the tower. Given the varying sensor station positions, the bluetooth antenna's range is important and something we had to consider when choosing antennas.

When choosing components for sensor stations, we had the goal of making the prototype compatible and unified, therefore the I2C protocol was chosen for the communication between integrated circuits (ICs) or electronic components within the system. This can provide a better flow for the development process and lead to a more efficient and effective system.

The sensor stations use the same bluetooth module and antennas as the gateway, together they (module and antenna) will play the role of a transmitter, sending the

measurements to the gateway. All of them are also equipped with the Ruuvi 1000 mAh Li/MnO2 CR2477T battery, which can operate in temperature between -40°C and 85°C.

3.2.1. Color sensor station

As for the color sensor chip, the TCS3200 color sensor with the a TAOS TCS3200 RGB sensor chip (a programmable color light-to-frequency converters) and 4 white LEDs was chosen for the the prototype on breadboard. However, later we found out that the chip itself was old and no longer available on the market. We search for another sensor, the TAOS TCS3470, a color light-to-digital converters with IF filter fits our purpose, however, the same situation occurred, the sensor is no longer manufactured. Alternatively, we found the VEML3328. The purpose of the VEML3328 sensor is to detect various types of light, including red, green, blue, clear, and infrared, using integrated photodiodes and circuits on a single CMOS chip. Its low power consumption and wide operating temperature range (-40 °C to +85 °C) make it ideal for this project.

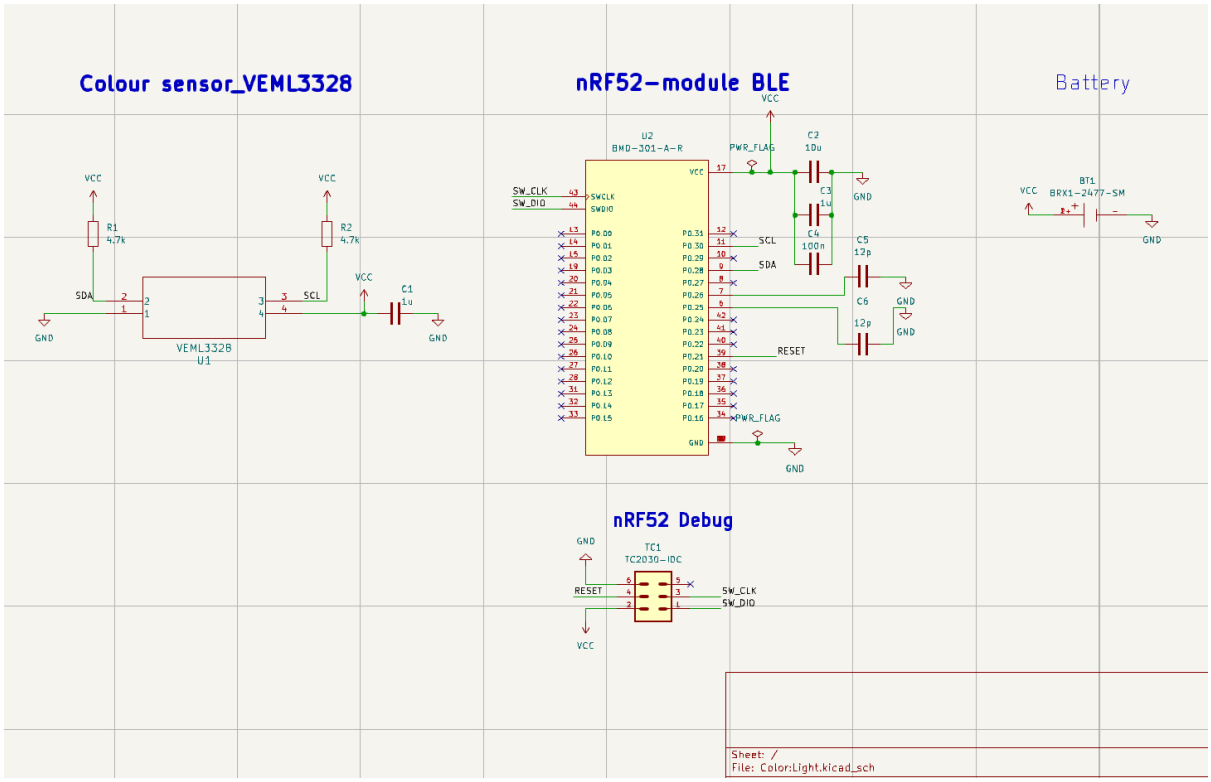


Figure 6: Schematic of Color sensor

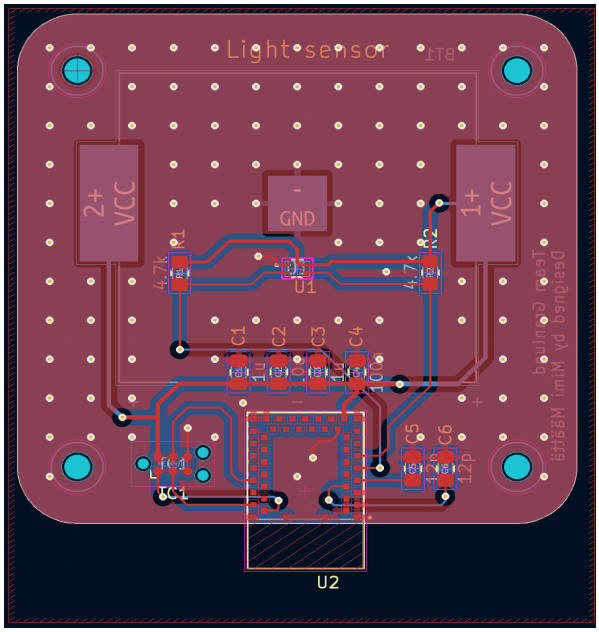


Figure 7: PCB design of Color sensor

3.2.2. Temperature sensor station

The initial plan was to use thermocouple for a better accuracy and attachment on the mental where we want to measure the temperature. However, due to the availability and price range of the thermocouple, we decided to just use a simple digital temperature sensor, MCP9808 which is developed by Microchip Technology Inc. Its primary purpose is to sense temperatures between -20°C and +100°C with an accuracy of ±0.25°C/±0.5°C (typical/maximum). The sensor operates on a voltage range of 2.7V-5.5V and has an operating current of 200 µA (typical).

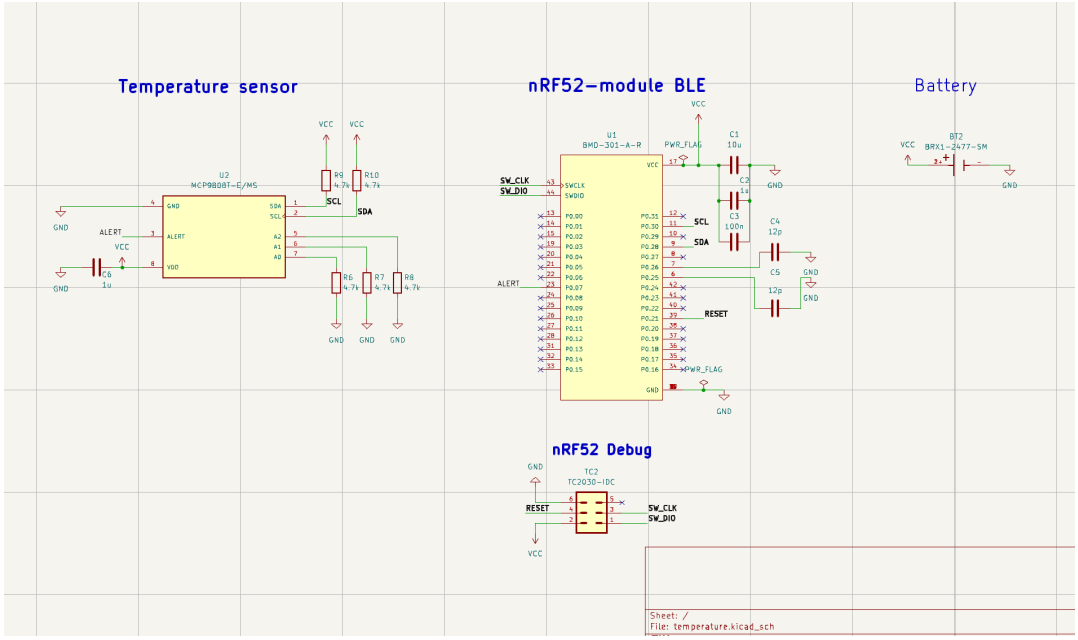


Figure 8: Schematic of Temperature sensor

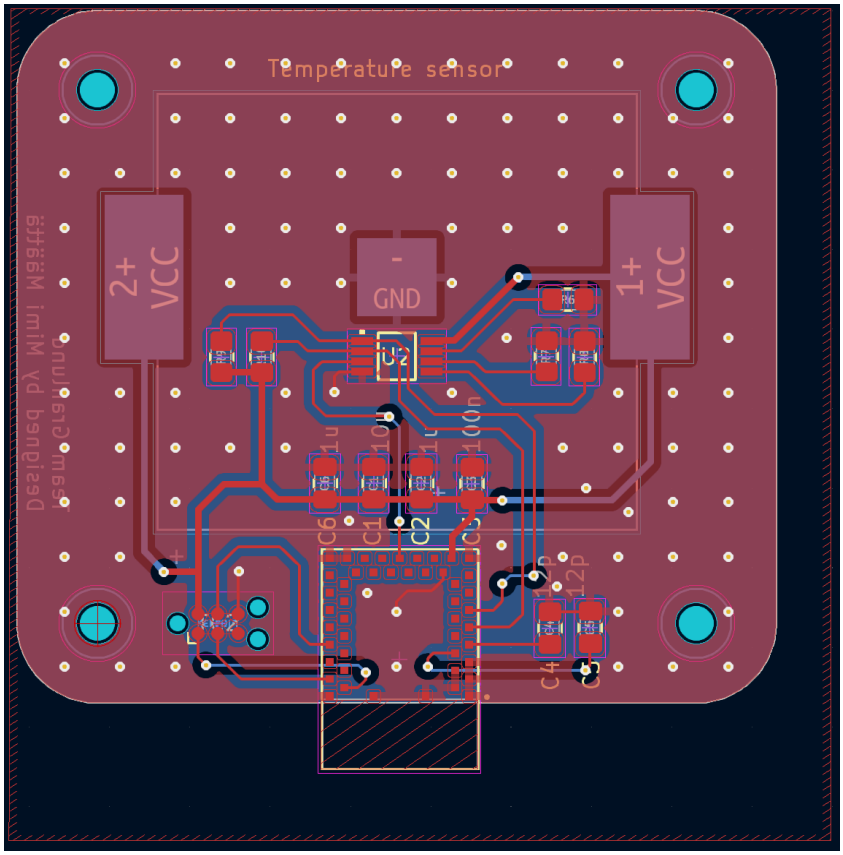


Figure 9: PCB design of Tempurature sensor

3.2.2. Accelerometer sensor station

The ICM-20948 is a 9-axis MotionTracking device designed for low power consumption. It features a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis compass with a wide range up to ±4900 μT. In another terms, this is a small but powerful sensor that can detect various types of movement. It can sense the tilt and the turn of the tower, as well as magnetic fields.

While designing the circuit for the Accelerometer sensor station, it came into knowledge that the Digital I/O supply voltage (VDDIO) only ranges 1.7-1.95V, so the voltage regulator was implemented into the schematic, which is shown in the image below.

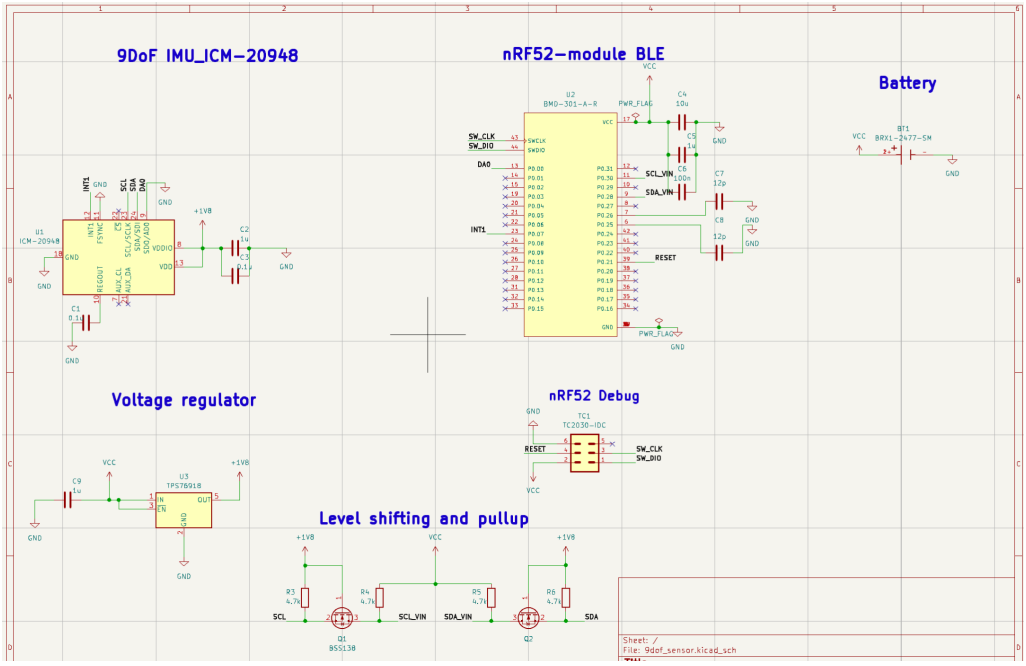


Figure 10: Schematic for Accelerometer sensor

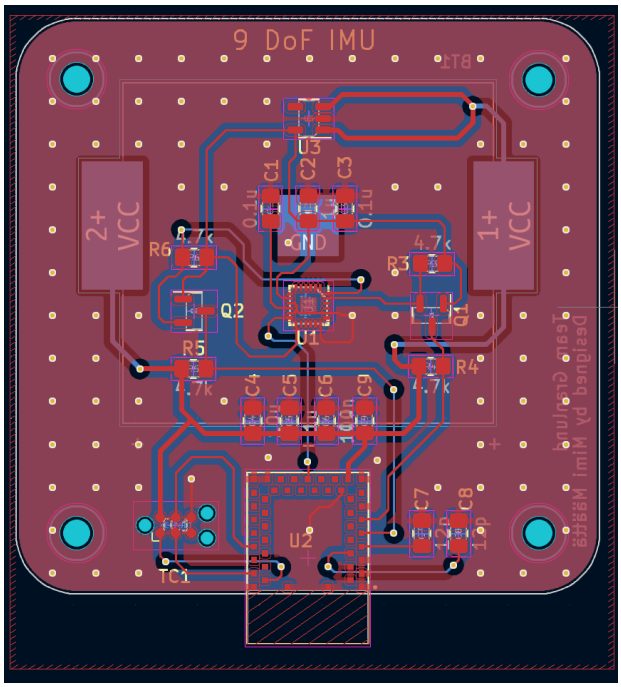


Figure 11: PCB design for Accelerometer sensor

3.3 Hardware assembling

The assembling faced challenges, first of all there was the delay of component delivery. Then, some of the components were extremely small that was hard to place on the pcbs. Also, due to the size and light weight of the components, they faced the risk of flying away in the reflow oven. The image below shows the final product, on the

top row, left to right in order is Color sensor, Accelerometer sensor and Temperature sensor; on the bottom row, on the left is the gateway and on the right it shows how the back of the sensors look like with the CR2477T battery attached.

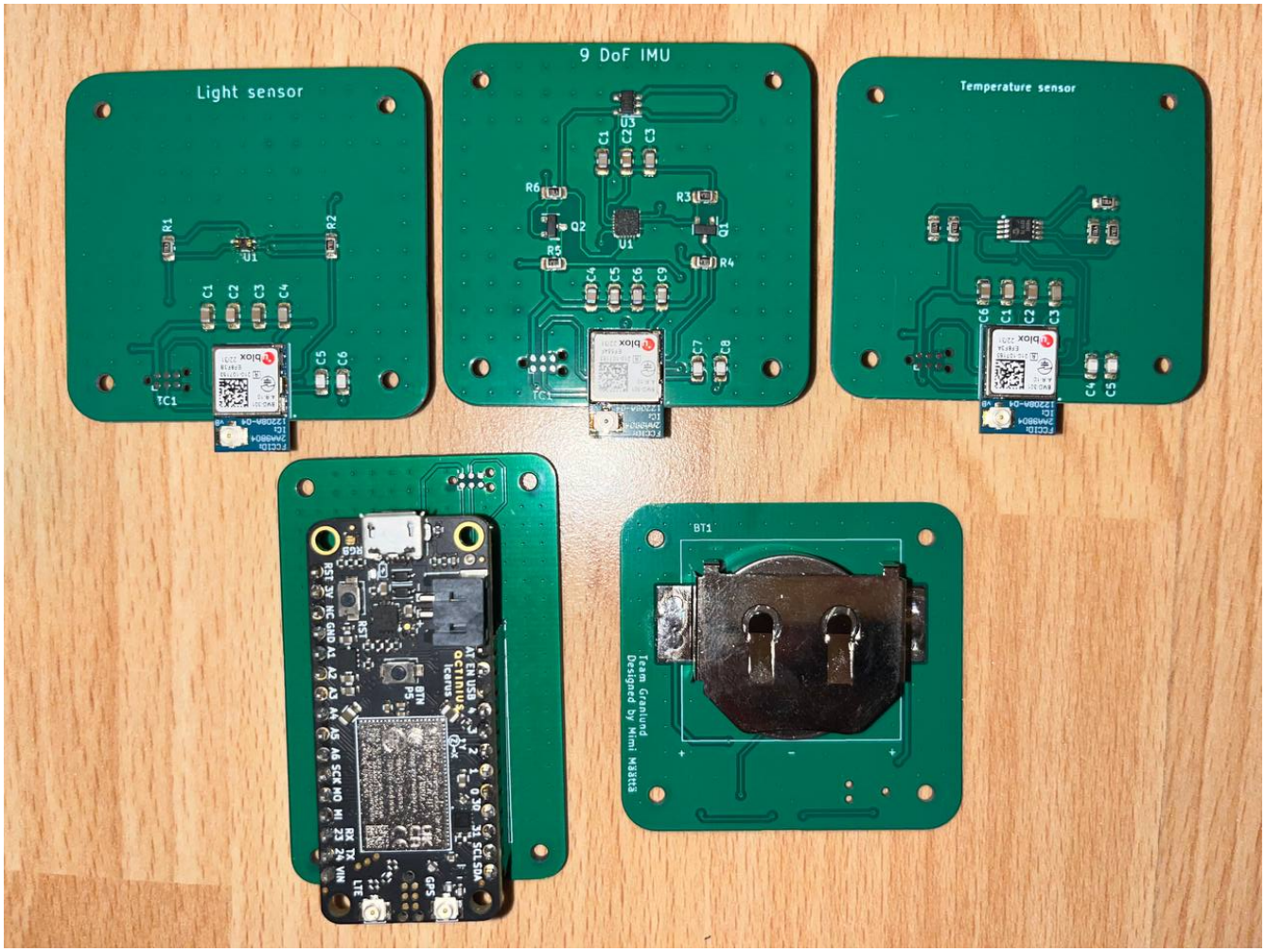


Figure 12: All PCBs after assembled

4. Data gathering and transferring

4.1 Icarus IoT Board

4.1.1 Example of data sent

First version

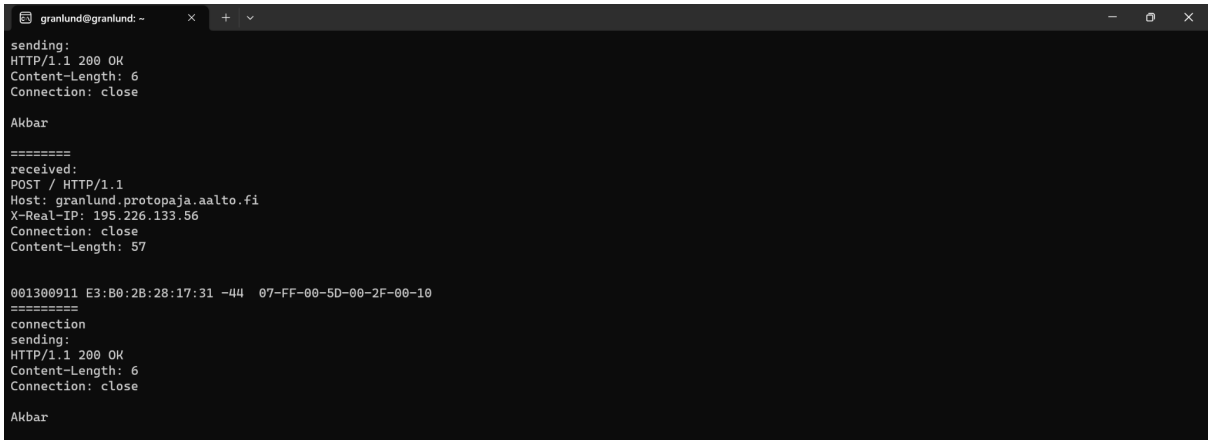


Figure 13: First version of data sent on Icarus board

Each BLE device has their own unique MAC address, to know which sensor data is it, E3:B0:2B:28:17:31 was address for device that had accelerometer in it. -44 is RSSI and the last element is the sensor data in hex format.

Second version

It was agreed that the gateway will not filter the beacons by their MAC address and will forward every beacon it listens to. This means any bluetooth device that advertises data will be sent to the server and the server would filter them. This deviated from the original plan where the gateway would collect the sensor data for a certain period of time and send many measurements at once to the server in order to minimise battery consumption.

Below is the an example of data sent:

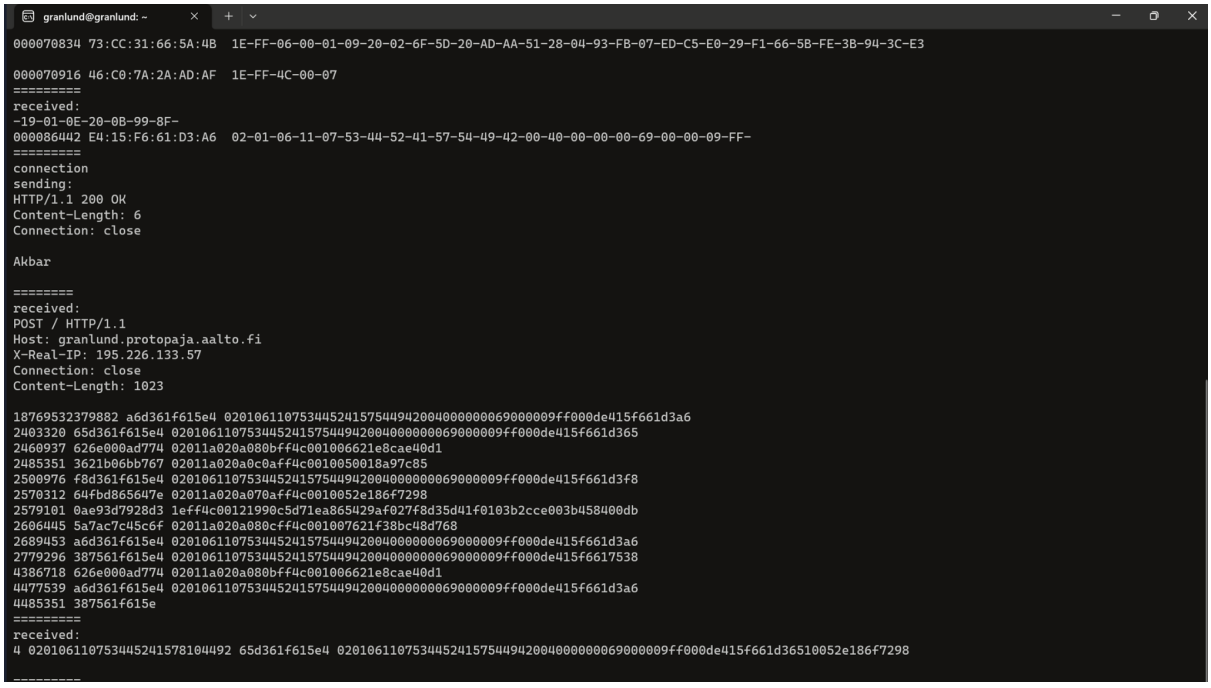


Figure 14: Second version of data sent on Icarus board

The structure of the message is following:

“Timestamp in microseconds” “ MAC address” “Advertising payload data in HEX format (where the sensor data is stored)”

4.1.2 HTPPs protocol

For data transfer from Icarus to the web server we used HTTPs requests, we used POST requests with a payload as UART message. For this purpose we used NRF Connect SDK v2.2.0 https_client sample and also echo_bot sample for UART readings. This is inside Icarus, below is the sample of making http post request with temperature data with its address. There is also a response from the server 200 meaning the server has successfully received it. Picture below shows Icarus’s serial monitor operating the post requests:

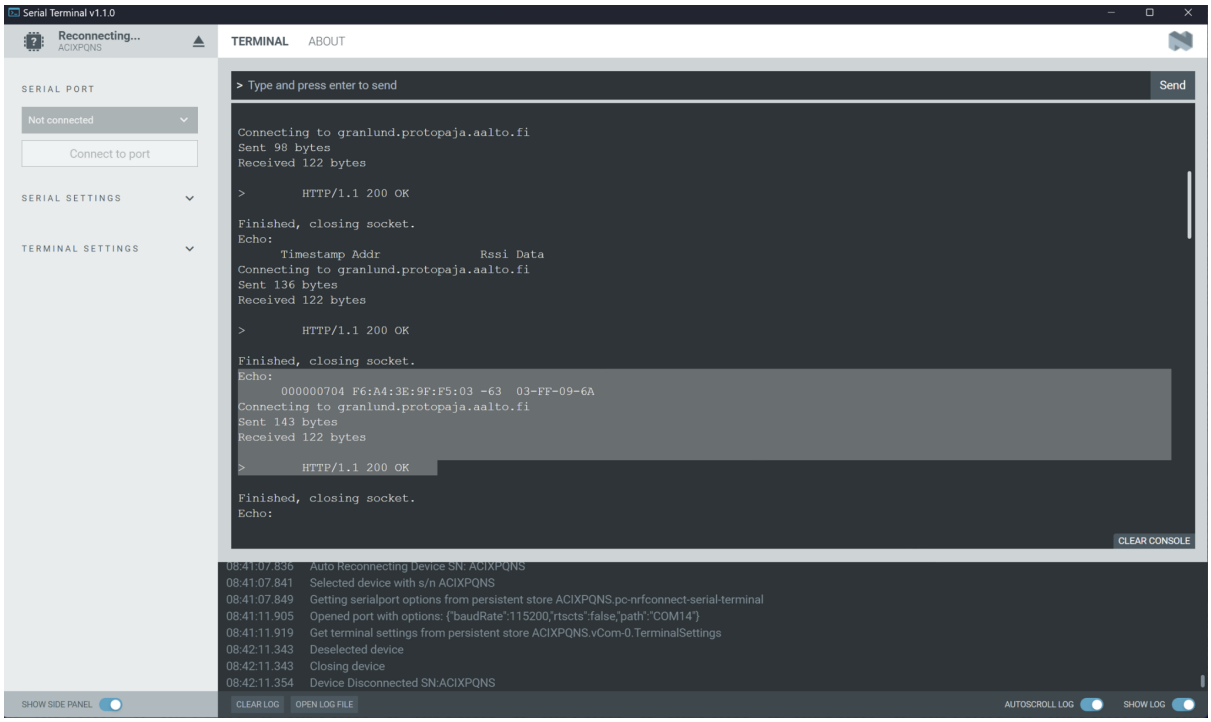


Figure 15: Icarus’s serial monitor operating the post requests

4.1.3 Setting up and configuration for successful build process

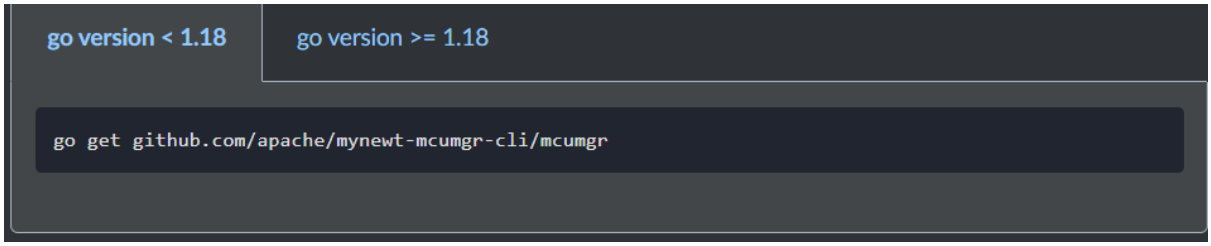
In order to be able to flash the code to the Icarus board with only the USB-connector, the CONFIG_BOOTLOADER_MCUBOOT=Y should be added in the prj.conf file as shown below:

```
CONFIG_NRF_MODEM_LIB=y

CONFIG_NETWORKING=y
CONFIG_NET_SOCKETS=y
CONFIG_NET_SOCKETS_POSIX_NAMES=y
CONFIG_NET_NATIVE=n
#UART
CONFIG_SERIAL=y
CONFIG_UART_INTERRUPT_DRIVEN=y

CONFIG_HEAP_MEM_POOL_SIZE=1024
CONFIG_MAIN_STACK_SIZE=2048

CONFIG_MODEM_KEY_MGMT=y
CONFIG_LTE_LINK_CONTROL=y
CONFIG_LTE_AUTO_INIT_AND_CONNECT=n

CONFIG_NEWLIB_LIBC=y
CONFIG_BOOTLOADER_MCUBOOT=y
```

Figure 16: configuration to flash the Icarus board

This configuration addition enables the building process of the Icarus IoT board to create a binary file ”app_update.bin” which is needed for flashing commands. If this config addition is not added, then the building process will generate only merged.hex or zephyr.hex files that are not flashable through the USB-connector.

After building process in VScode, mcumgr utility is downloaded via Go language:



mcumgr utility is used to flash the built binary file from building process as follows:

mcumgr image upload app_update.bin --conntype=serial --connstring="dev=COM14,baud=115200"

Before entering this command the Icarus IoT board should be in a bootloader mode. Bootloader mode is entered by this sequence:

1. *Hold reset button*
2. *While holding the reset button, press and release the user button.*
3. *Release the reset button*

Now the icarus IoT board is in bootloader mode, meaning that it is ready to update its code.

The above mentioned command is entered in the terminal inside <path-to-sample-dir>/build/zephyr/ directory. Below is shown the successful build that generated the needed binary file and also how to flash the code.

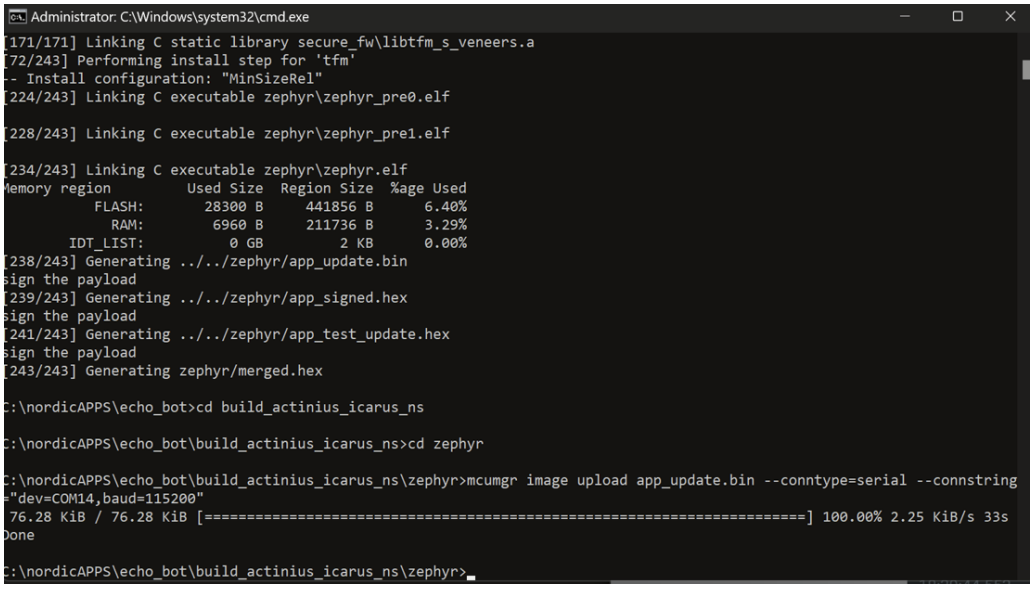
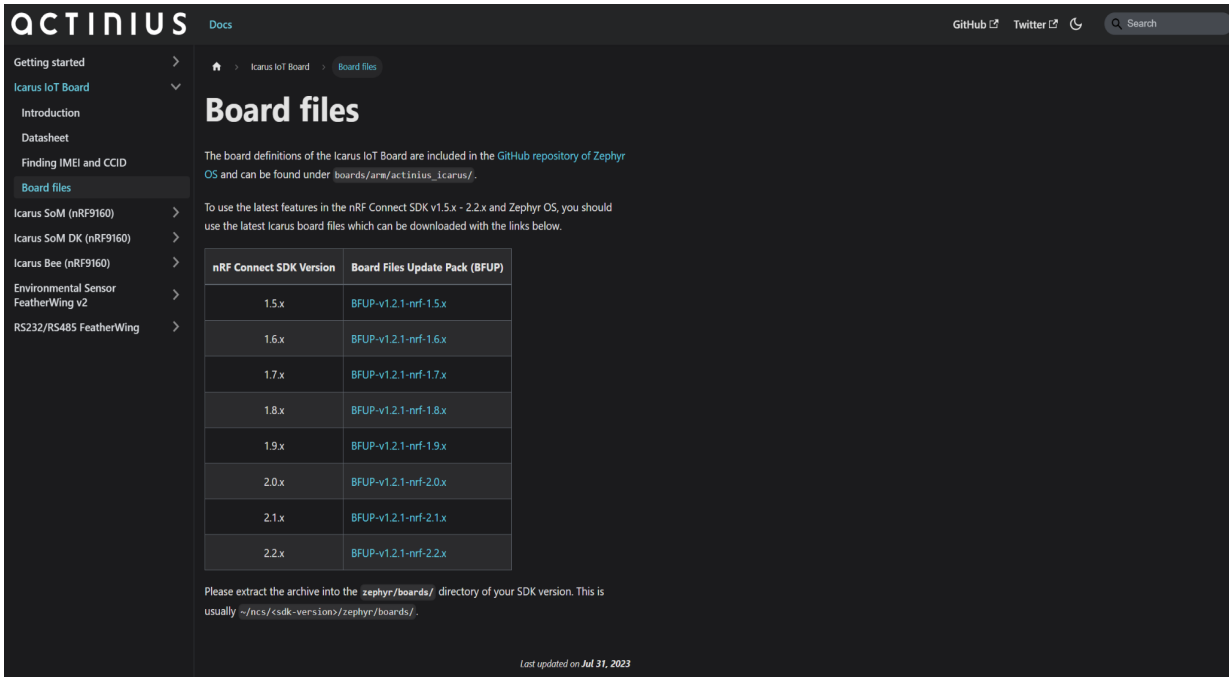
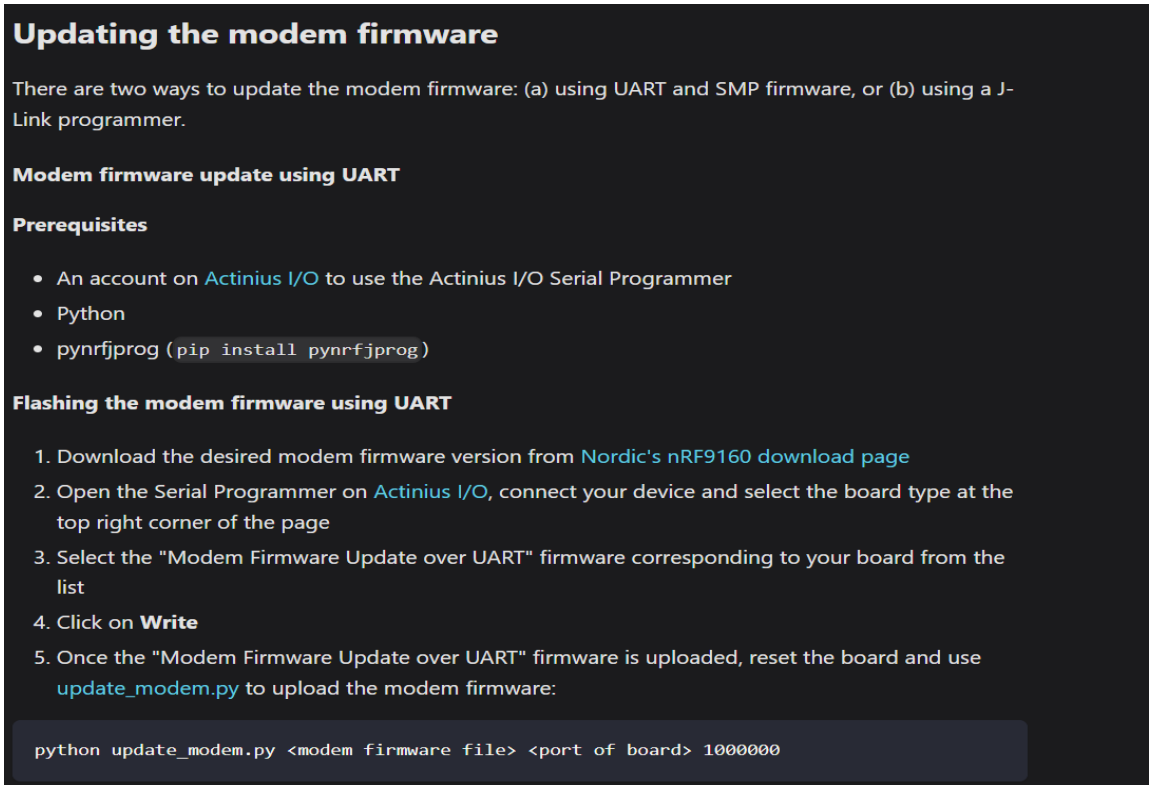


Figure 17: the successful build that generated the needed binary file and also how to flash the code

NRF connect sdk 2.2.0 was used to program the Actinius board. As advised with Actinius support, in the downloaded nrf sdk inside “C:\ncs\v2.2.0\zephyr\boards\arm” directory where the board files are located, the existing board files from SDK for actinius icarus was replaced with board files downloaded from icarus’s website.



Modem firmware was updated to 1.3.5 using UART as instructed below:



Updating the modem firmware

There are two ways to update the modem firmware: (a) using UART and SMP firmware, or (b) using a J-Link programmer.

Modem firmware update using UART

Prerequisites

- An account on Actinius I/O to use the Actinius I/O Serial Programmer
- Python
- pynrfjprog (`pip install pynrfjprog`)

Flashing the modem firmware using UART

1. Download the desired modem firmware version from Nordic's nRF9160 download page
2. Open the Serial Programmer on Actinius I/O, connect your device and select the board type at the top right corner of the page
3. Select the "Modem Firmware Update over UART" firmware corresponding to your board from the list
4. Click on **Write**
5. Once the "Modem Firmware Update over UART" firmware is uploaded, reset the board and use update_modem.py to upload the modem firmware:

```
python update_modem.py <modem firmware file> <port of board> 1000000
```

Figure 18 Instruction for updating Modem firmware to 1.3.5 using UART

4.1.4 Confusions with community support and Actinius support

In the very beginning flashing the code was not possible with MCUBOOT, building process failure. we reconfigured Kconfig.deconfig file to add the TFM support and also made pristine build. The icarus became programmable. Later actinius support advised us that the SDK's board files for actinius board needed to be replaced with board files provided by actinius's website.

NCS 2.4.0, 2.3.0 did not work for icarus IoT board. One of the possible reasons is that its modem firmware was updated late, after we chose 2.2.0.

Actinius support provided us a device tree overlay file and prj.conf file in order to configure UART, but it was not needed.

Apparently it was a problem of zephyr's device binding, instead of DT_CHOSEN we used DT_NODELABEL in the main code.

http sample provided by ncs 2.2.0 for some reason we don't know yet is not buildable due to KCONFIG issues. Actinius support responded to this issue by suggesting using another sample. Below is the error message when building the http sample:

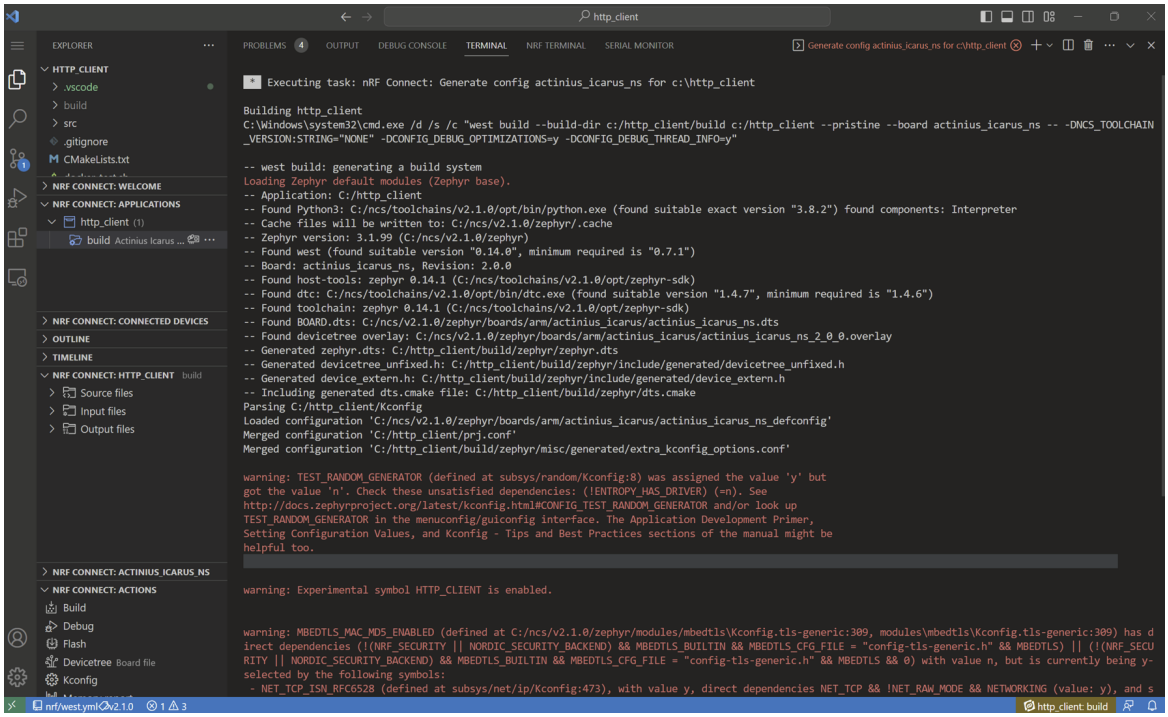


Figure 19: Error message when building the http sample

NCS's https_client sample is buildable, and this was chosen for the project.

From NRF devzone, Peripheral_UART and central_UART were suggested samples to use for BLE part, they are not suitable for our project as it is a pair connection type of BLE and it allows only 2 devices to transfer data. In our case we have 4 BLE devices, one of which is the central that listens to the broadcasters. So we used Adafruit's BLE beacons which broadcast data independently without connection to another device, and Adafruit's BLE central that listens to the advertising packets.

4.2 Bluetooth low energy devices

4.2.1 Sensor station advertising

BLE advertising is written in Arduino that reads data from a sensor, and broadcasts it as a bluetooth beacon. The code uses the Adafruit Bluefruit Library for BLE functionality.

4.2.2 Central device

BLE beacons scanning is written in Arduino with the help of Adafruit's Bluefruit library.

First version

The gateway's nrf52 scans and filters the devices by their MAC address, the provided picture shows the scanned sensor station: E3 address is the accelerometer data and the F6 is the temp sensor data.

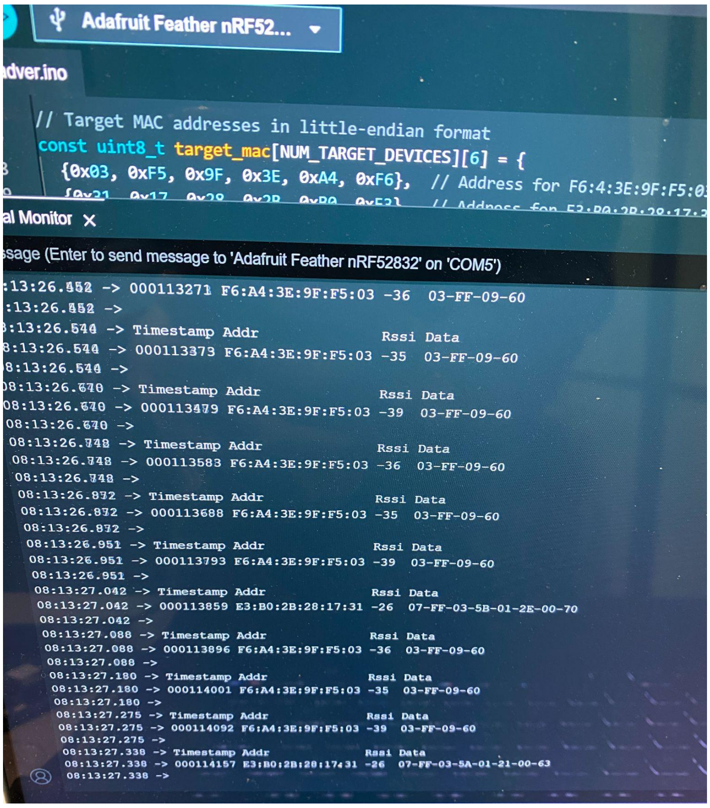


Figure 20: The scanned sensor station

It forwards the scanned data to UART which then Icarus IoT board makes a HTTP post request with the given UART buffer.

Second version

Gateway's nRf52 is listening to every beacon that it can scan within the range.

Figure 21: Listening to beacons in the gateway

5. Data processing and data display

5.1. Software architecture & server specifications

The following figure shows a block diagram of the overall software architecture used on the server and data processing side of this project. The light blue boxes describe the four main parts of the structure (cloud server, database, website and data processing scripts). The project itself is currently hosted on Aalto's cloud unix server, granlund@granlund.protopaja.aalto.fi. The web application can be found through the domain: https://granlund.protopaja.aalto.fi/, which has been set as a secure website.

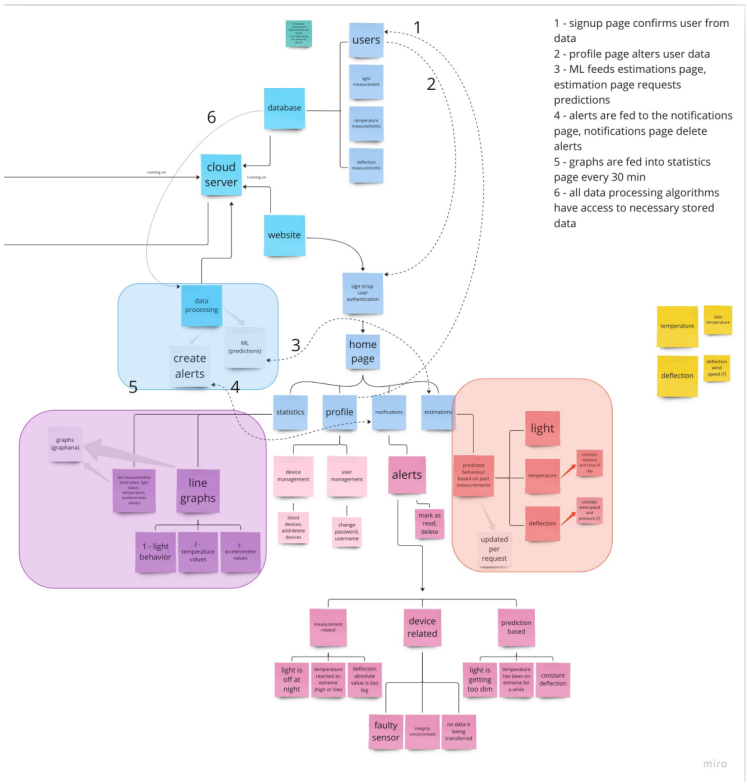


Figure 22: Block diagram of overall software architecture

Ideally, the website is set to provide a smooth and informative user experience. Due to time constraints, our focus is going to be specifically on gathered data display, notification alerts regarding data points, and display of predictions based on the measurements gathered by our device. Other additional features such as user authentication, profile and customizations will be left on the back burner for possible future iterations.

The data flow on the server follows the following steps:

1. HTTP requests is actively listening for data points (which will be received every five minutes);

2. Data, received as a JSON file, is transferred to our database with the help of a python script;
3. Backend scripts analyse new data for anomalies and activate notifications as required;
4. Data is then updated on the website graphs;
5. Prediction calculations are rerun and also updated on the website.

The website is run on a flask app, which is started by running a python script on the server. This can be done from specific computers with ssh key access to the server. This app can be running continuously so that the website is always online. The main language when coding flask apps is python, but for the website html, cs and javascript are also used.

5.2 Data storing

The data is stored on a Postgres hosted database. This has been installed on the project server. The project currently consists of a single database, protodata, with two tables (measurements and alerts). The following table shows the columns and data types used for the measurements table, which is used for storing all data points received:

Column name:	**Data type:**
m_id	Integral (primary key)
date	String
timestamp	String
address (of sensor station)	String
measurement	Float

Table 1: Column name and data types of all data points received

The alerts table has reference to the measurements table's primary key values. This table stores the alert messages created for any data anomalies found amongst the measurements. The columns and data types are as follows:

Column name:	**Data type:**
a_id	Integral (primary key)
alert	String
type	String

measurement	Integral, reference to Measurement's m_id

Table 2: Data names and types for the alert messages

The server is constantly listening for HTTP requests from the gateway. Our original plan and agreement was that the data would be sent from gateway to server in a JSON file format. However, due to time constraints, that was not achieved. The data is received from the gateway as plain text strings. A quick solution to this problem was to create a script on the server that would convert such strings into JSON files, which allows the received data to be uploaded into the flask app and displayed to the user. From the JSON file, the running python script then correctly allocates the data into our database. Every time data is received, a new JSON file is created with data points (measurements) gathered from our sensors by the gateway. The JSON template can be found on the appendix of this report.

5.3 Code structure

The server code is organised on the following structure:

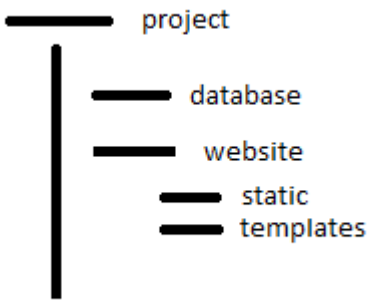


Figure 23: Organised structure of server code

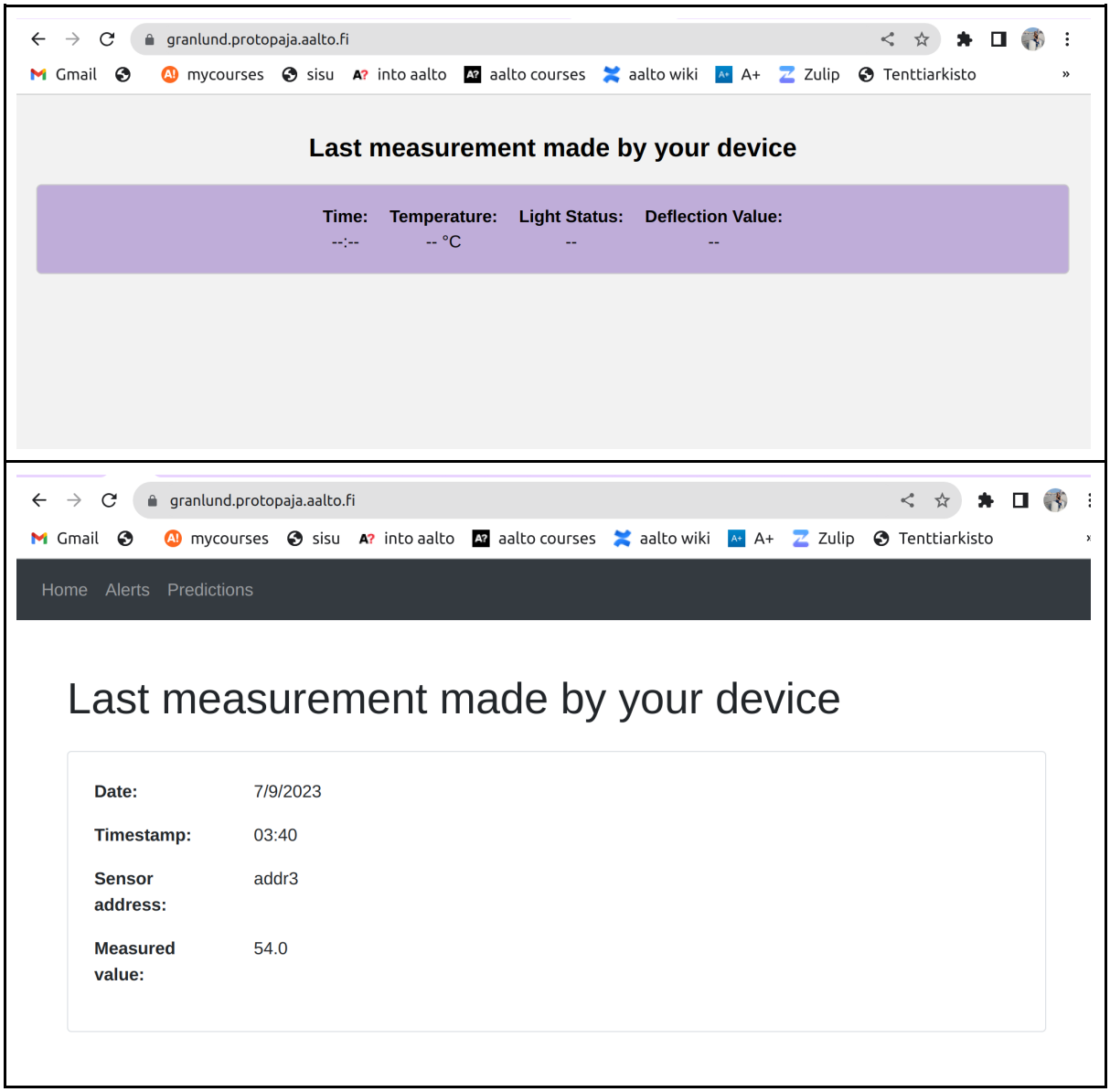
The main directory, project, contains all the necessary files involved in data storage and the web application. In it, the database and website directories are found, as well as the main.py file responsible for running the application.

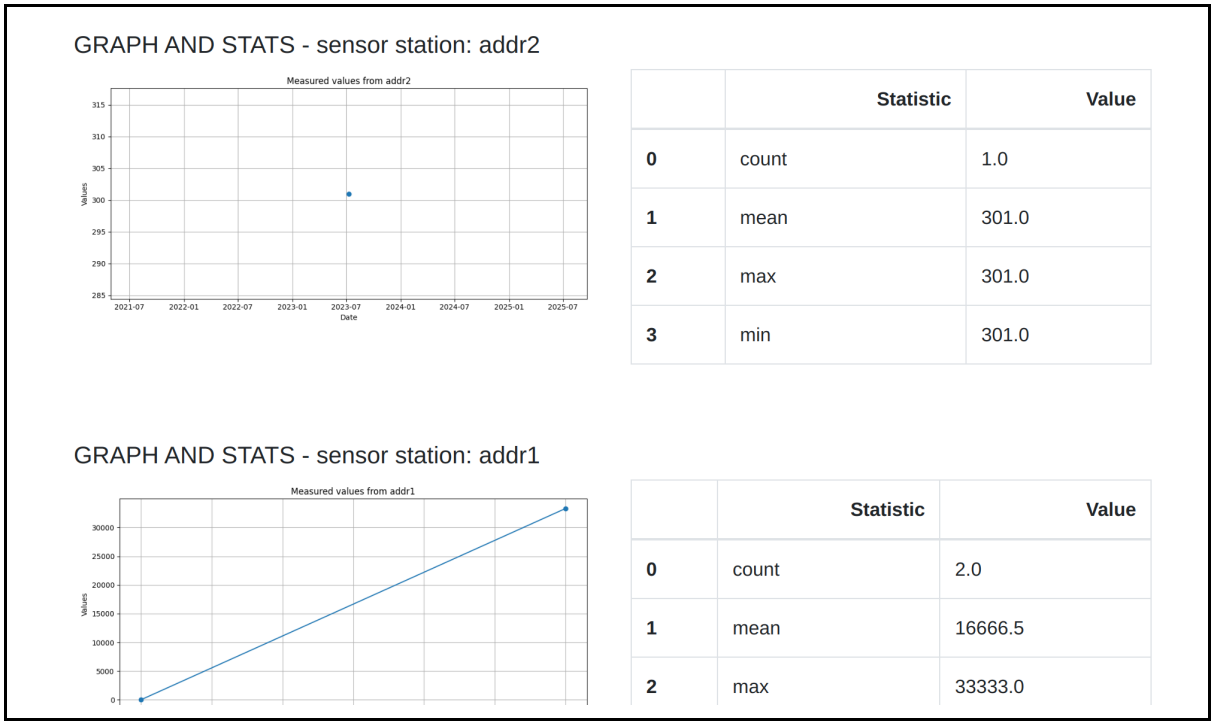
The database directory contains all the scripts related to creating the database and the tables. Whereas, the website directory handles the data display and our api. In it, you can find python scripts responsible for: the setup and handling of the website pages (views), the data processing and storage on database, as well as a script handling graph and table formatting. Additionally, the two folders inside the website directory, static and website, store graph pictures for display and html templates rendered from the views script, respectively.

For the website's css and javascript formatting, both custom code and Bootstrap library were used. All code can be found on our public github repository.

5.5 Website & notifications/alerts

The images below demonstrate the first and final iterations of the website's home page, respectively. The home page of the website displays information about the last known measured values, which is updated every time a new measurement is processed. Originally, the goal was to display the latest measurement made by each sensor station. However, due to last minute changes on the data transfer (from gateway to server) format, the newer website page displays the latest recorded value. The home page also displays graphs for each sensor station (present on the database) showing its past measurements and some statistics.





	Statistic	Value
0	count	1.0
1	mean	301.0
2	max	301.0
3	min	301.0

	Statistic	Value
0	count	2.0
1	mean	16666.5
2	max	33333.0

Figure 24: The first and final iterations of the website's home page, respectively

The graphs and statistics display works so that every single measurement made by a specific MAC address is collected and plotted on a graph over the time they were measured. The website will display as many graphs as there are different addresses registered on the database, i.e. sensor stations collecting different measurements.

In addition to the data display, our project aims to keep the user informed about measured anomalies. Our website will display such anomalies as notifications to the user. Therefore, the website also has an "alerts" page, shown in the image below, which shows users of any unexpected measurement made on that day or previous dates.

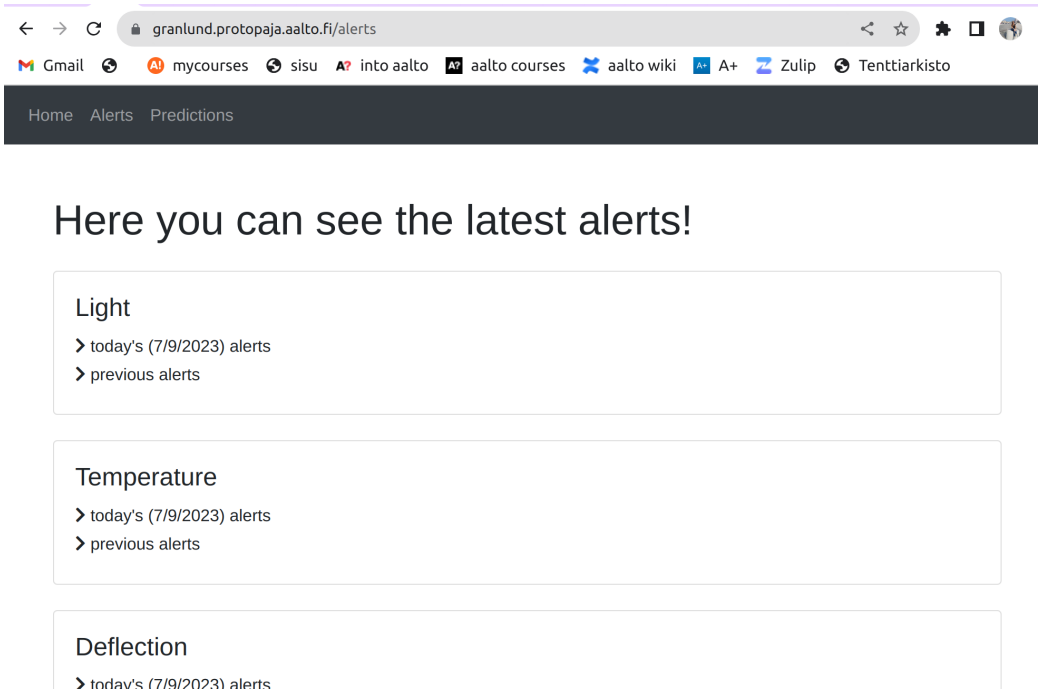


Figure 25: Alerts page from the website

The user will be alerted given the following scenarios:

1. the tower light is too dim at night time,
2. the temperature of the tower has reached an extreme value,
3. the measured deflection is constant,
4. the measured deflection is too extreme,

Lastly, our website has a third separate page reserved for displaying maintenance predictions based on the gathered data. However, due to insufficient gathered data, limited knowledge and understanding of the physics involved and our data being limited to only a few sensor stations, we were not able to create accurate and significant predictive algorithms. Currently, it contains placeholder text and some overview comments on the tower structure.

6. Reflection of the Project

6.1. Reaching objective

Our simplified goal with this project was to create a health monitoring system including individual sensor stations, a gateway and a data display application. This basic target was achieved and we finalised a working prototype. In technical terms, we had a few shortcomings that are further described in the following sessions. Most of our adjustments were made due to our limited skills and the overall time taken to complete each phase.

6.2. Timetable

Our timeline and schedule significantly deviated from our original plan, mostly due to external working responsibilities of multiple team members, as well as project tasks taking more hours than expected. Regardless of the schedule, most main features were completed, especially from the hardware and server sides of the project.

We found it quite hard to keep up with our planned milestones given our own working hours and different ways of completing tasks.

6.3. Risk analysis / Riskianalyysi

In the project plan, risks were identified as follow:

Risk	Management
time management and scheduling	create a detailed task and timeline plan
shipping delays of components	order the crucial components in advance
unavailability of needed components	research alternatives
lack of knowledge -> additional time consumption	overestimate time for tasks rather than underestimate
safety and dangerous situations when prototyping with hardware and physical tools	properly learn safety procedures ahead of time, take caution and wear appropriate gear/clothing
Personal safety measurements and risks at test site locations	know and follow all safety procedures
Hardware safety during tests	analyze material risks and potential hardware problems when testing device

A lot of problems encountered throughout the project, as well as the factors contributing to the delays were documented in the project plan.

The availability of components problems were evaluated as a potential risk. To illustrate, an incorrect order for antenna cable adapters was placed, which led to a delay in connection testing. Furthermore, the final order with all the necessary components for assembling the PCB was facing shipping delay, which caused a big delay on the assembling the devices.

The plan had mentioned few points about safety and dangerous situation as risks. However, none of those situations had occurred. All safety procedure was applied while working accordingly.

It is worth mentioning that, even though the risks were well assessed, we left out the considerations related to communication challenges, unforeseen natural events like illnesses, and issues pertaining to time management. These factors should also be recognized as potential risks.

6.4. Shortcomings with the data transferring process

Whilst there is a working code that sends data from the sensor stations, to the gateway and to the server, this code was tested with the components using a breadboard and not uploaded to the PCB. In future iterations, this would be the first step to improving the prototype. Furthermore, we had originally planned that the gateway would gather measurements until a byte is full and only then send the data collectively to the server in order to have lower power consumption. However, this is currently not happening as the gateway sends the measurements to the server as soon as they are received from the sensor stations.

6.5. Project simplification

Due to time and skills constraints, we also had to simplify or give up on some of our original additional features. From the data analysis and future predictions point of view, we found it hard to create predicting algorithms that would give up significant values or results. When creating future predictions, we would have needed an extensive dataset created by our device in order to get a better understanding of what the measurements would look like and what conclusions could be taken from them. Furthermore, the implemented sensor stations do not provide enough data to give conclusive predictions about malfunctions and maintenance timeframes. For example, there are multiple factors influencing the deflection of the tower, and data from multiple sensors (temperature, tower movement, material weight, wind pressure) could be combined for a possible prediction of future deflection values. Our current knowledge level of tower physics is not enough in order to create such a predictive algorithm.

The other unfortunate feature we had to forgo was utilising energy harvesting techniques. We did not have enough time to implement this feature after creating the basic prototype. Given our research on the topic and planning, we would have liked to experiment with solar panels as an energy harvesting technique to at least prolong the battery life of our gateway and sensor stations. This improvement could be a future direction of this prototype.

7. Discussion and Conclusions

7.1 Personal experience

Beatriz Glaser:
I am satisfied with my work on this project. I have been looking for a chance to further my software skills, and being responsible for the server-side processes (data processing & display) did exactly that. I had never created a flask app before, worked with protocol requests nor ever accessed a remote server before. This experience taught me a lot, both hard and soft skills, that are necessary in the development of projects. It was especially valuable to work on a project given by a company, looking for solutions to a real current problem.

Mimi Määttä:
Overall, the course has provided me with a deeply educational experience. My main focus on the project was Hardware. Although I had had a couple of prototyping courses as a part of my major, I had only touched the surface of circuit design. Furthermore, my knowledge of different sensors and components was limited, and at the same time, creating and assembling PCBs was completely new to me. Throughout this course, I got to work more with Kicad software, and learnt to design efficient PCBs. I saw myself improve tremendously in this area.

7.2 Conclusion

There was a lot to learn from this project from the perspective of all team members. Creating it from scratch (brainstorming, conceptualising, planning and developing) was a hard challenge to overcome, however we succeeded at the end delivering a working prototype.

List of Appendixes

Appendix 1: Project plan (PDF)

Appendix 2: Datasheet for Icarus IoT Board V2, https://docs.actinius.com/icarus/datasheet

Appendix 3: Datasheet for BMD-301, https://content.u-blox.com/sites/default/files/BMD-301_DataSheet_UBX-19033351.pdf

Appendix 4: Datasheet for VEML3328, https://www.vishay.com/docs/84968/veml3328.pdf

Appendix 5: Datasheet for MCP9808, https://ww1.microchip.com/downloads/en/DeviceDoc/MCP9808-0.5C-Maximum-Accuracy-Digital-Temperature-Sensor-Data-Sheet-DS20005095B.pdf

Appendix 6: Datasheet for ICM20948, https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf?ref_disty=digikey

References / Lähteet

[1] Actinus, Icarus IoT Board V2,
https://docs.actinius.com/icarus/introduction/

[2] Adafruit, Bluefruit nRF52 Feather Learning Guide,
https://learn.adafruit.com/bluefruit-nrf52-feather-learning-guide/arduino-bsp-setup

[3] Nordic Semiconductor, Devzone, https://devzone.nordicsemi.com/

[4] PostgreSQL, https://www.postgresql.org/

[5] Psycopg, Psycopg – PostgreSQL database adapter for Python,
https://www.psycopg.org/docs/

[6] U-Blox, BMD-301 data sheet, https://content.u-blox.com/sites/default/files/BMD-301_DataSheet_UBX-19033351.pdf

[7] Zephyr, https://docs.zephyrproject.org/apidoc/latest/index.html