OULUN AMMATTIKORKEAKOULU

DESIGN AND IMPLEMENTATION OF A MODULAR THREAD-BASED SENSOR PLATFORM WITH CLOUD INTEGRATION

Tuovi Littow & Marko Rautiainen
Bachelor's Thesis
Spring 2025
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology
Option of Device and Product Design

Authors: Tuovi Littow & Marko Rautiainen
Title of thesis: Design and Implementation of a Modular Thread-Based Sensor Platform with Cloud Integration
Thesis supervisor: Pasi Mustonen
Term and year of completion: Spring 2025
Pages: 63 + 1 appendix

The goal of this thesis was to design and implement a sensor platform that utilizes a Thread network for communication within a local network and then connects to the internet to transmit its data. The overall system consists of three core components: the sensor platform, the Thread Border Router, and the web application.

In order to gain a deeper understanding of the topic the work on this thesis began by studying the Thread protocol and comparing it to other wireless communication technologies commonly used in smart home systems.

In the second phase, a sensor platform was designed and implemented to collect environmental data. A web application was also developed to display and manage the sensor platforms and the data they collect. The sensor platforms were connected through a Thread mesh network, with a Thread Border Router used to enable communication between the local Thread network and the cloud-hosted web application.

Finally, the system was tested against the defined requirements to evaluate its functionality and performance. Additional tests were also conducted to test the behaviour of the Thread network. In the end, the system was also deployed and tested in a real-life use case scenario as a houseplant monitoring system.

As a result, the system met all the mandatory requirements. The sensor platform, Thread Border Router (TBR), and web application worked together as intended, enabling reliable data transmission from the sensors to the cloud. Testing of the Thread network also demonstrated its self-organizing and self-healing capabilities, exceeding expectations. Although some optional features, such as Over-the-Air updates and alert notifications, were not fully implemented, the system provides a good foundation for future development.

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Laite- ja tuotesuunnittelun suuntautumisvaihtoehto

Tekijät: Tuovi Littow & Marko Rautiainen
Opinnäytetyön otsikko: Modulaarisen Thread-pohjaisen sensorialustan ja pilvi-integraation suunnittelu ja toteutus
Työn ohjaaja: Pasi Mustonen
Työn valmistumislukukausi ja -vuosi: kevät 2025
Sivumäärä: 63 + 1 liite

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa sensorialusta, joka käyttää Thread-verkkoa paikallisessa viestinnässä ja yhdistyy sitten internettiin tiedonsiirtoa varten. Järjestelmä koostuu kolmesta keskeisestä osasta: sensorialustasta, Thread-reunareitittimestä ja verkkosovelluksesta.

Työ alkoi perehtymällä Thread-protokollan ominaisuuksiin ja vertailemalla sitä muihin älykotijärjestelmissä yleisesti käytettyihin langattomiin viestintäteknologioihin, jotta saataisiin syvällisempi ymmärrys aiheesta.

Toisessa vaiheessa suunniteltiin ja toteutettiin sensorialusta mittaamaan ympäristöolosuhteita. Samalla kehitettiin verkkosovellus, jolla voidaan hallita sensorialustoja ja visualisoida niiden keräämää dataa. Sensorialustat yhdistettiin Thread-verkon kautta, ja Thread-reunareititintä käytettiin mahdollistamaan viestintä paikallisen Thread-verkon ja pilvipalvelimella olevan verkkosovelluksen välillä.

Lopuksi järjestelmää testattiin työssä ennalta määriteltyjen vaatimusten perusteella sen toiminnallisuuden ja suorituskyvyn arvioimiseksi. Lisäksi tehtiin erillisiä testejä Thread-verkon käyttäytymisen selvittämiseksi. Viimeiseksi järjestelmää testattiin käytännön sovelluksessa, jossa sitä käytettiin taimikasvatuksen olosuhteiden valvontaan.

Tulosten perusteella järjestelmä täytti kaikki pakolliset vaatimukset. Sensorialusta, Thread-reunareititin ja verkkosovellus toimivat yhdessä suunnitellusti, mahdollistaen luotettavan tiedonsiirron sensoreilta pilveen. Thread-verkon testaus osoitti sen itseorganisoituvat ja itsekorjautuvat ominaisuudet, jotka ylittivät odotukset. Vaikka joitakin valinnaisia ominaisuuksia, kuten langattomia ohjelmistopäivityksiä ja hälytysilmoituksia, ei ehditty toteuttaa täysimääräisesti, tarjoaa järjestelmä vahvan pohjan tulevaa kehitystyötä varten.

CONTENTS

GLOSSARY

ESP32	A feature-rich series of microcontroller units with integrated Wi-Fi and Bluetooth connectivity, designed for a wide-range of applications.
IDE	Integrated Development Environment
IoT	Internet of Things
Mesh network	A mesh network is a local area network where infrastructure nodes connect directly and non-hierarchically with many other nodes, working together to efficiently route data between devices.
OTA	Over-the-Air: a method of updating embedded device firmware wirelessly.
REST API	Representational State Transfer Application Programming Interface
TBR	Thread Border Router
Thread	A low-power, wireless mesh networking protocol designed for smart home and Internet of Things (IoT) devices.

1 INTRODUCTION

The goal of this thesis is to design and implement a modular Internet of Things (IoT) sensor system that enables users to connect their electronic projects to the internet and smart home systems using the modern Thread protocol. The number of Internet-enabled devices has increased steadily each year, and it is estimated that the number of IoT devices worldwide will grow from 18.8 billion in 2024 to 40 billion by 2030 (Kumar 2024).

The subject of our thesis was not commissioned by any organization but came purely from our interest in studying IoT devices and technologies. We were also motivated by our personal real-world use cases where a device that could connect home-built technology projects to the internet and to other smart home devices was needed.

In our studies, we have had courses related to wireless communication and hardware design, but with our thesis, we aim to delve deeper into these subjects and acquire better understanding of smart home devices and their protocols.

To gain real-world experience in addition to theoretical learning, one part of the project is designing and building a modular sensor platform that utilizes the future-proof Thread protocol for communication. Modularity allows users to add different types of sensors and thus adapt the sensor platform for various applications. As a practical example, we are using the sensor platform to build a monitoring system for houseplants.

Our project can be divided into three major parts: the sensor platform, the Thread Border Router (TBR), and the web application. The sensor platform consists of an ESP32-H2 microcontroller, a power source and connections for attaching sensors. In our use case, we are using a sensor that measures both temperature and humidity, and two soil moisture sensors to track the conditions of plants in real time. This system monitors key factors for optimal plant growth and ensures the best possible conditions for successful gardening.

The second part of the project is the Thread Border Router, which is used as an intermediary between the sensor platform and cloud, transmitting data between them.

The third part is the web application, in which sensor data can be visualized and monitored using graphical tools. Additionally, we aim to implement two-way communication, enabling us to both read sensor data and send updates to the sensor platform.

After completion of these three main components, we integrate them together and use it in a real-world use case to build a houseplant monitoring system. All the tasks mentioned above are planned to be completed over a total duration of 10 weeks.

2 THREAD PROTOCOL AND COMPARISON

2.1 Thread protocol

Thread is a low power mesh networking technology for IoT (Internet of Things) products. It was designed for smart home systems and commercial applications. The Thread network is based on the IEEE 802.15.4 standard, and it allows devices to communicate simply, securely and with low energy consumption. (Thread Group 2025.) The Thread Network consists of different parts that are shown in Figure 1 and will be explained in more detail below.

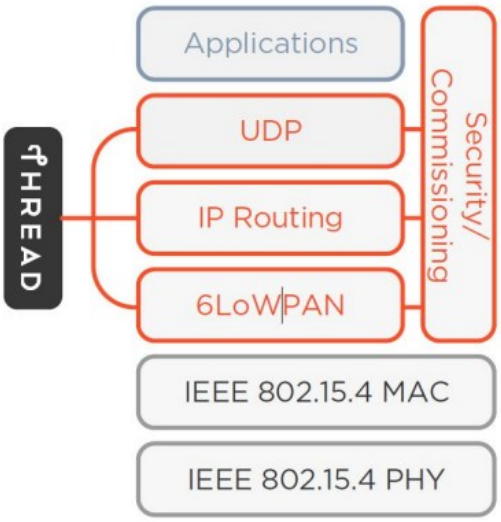

FIGURE 1. Overview of Thread Specification (Thread Group 2020)

IEEE 802.15.4.

The IEEE 802.15.4. is a radio standard that provides the lower network layers for energy-efficient wireless personal area networks (WPAN). The standard includes a physical layer (PHY) and a medium access control layer (MAC). The MAC layer is used as an interface between the physical layer and the chosen application system. (Poole.)

6LoWPAN (Low-Power Wireless Personal Area Networks)

The 6LoWPAN network is a low power mesh network which was built so that even the smallest devices with limited processing capabilities could be part of the IoT. The 6LoWPAN network offers end-to-end communication where each node has its own IPv6 address. In Thread devices, the 6LoWPAN network is used for sending IPv6 packages over the IEEE 802.15.4. networks. (GeeksforGeeks 2024.)

IP Routing

IP-Routing defines the shortest path for the data travelling through a network. (GeeksforGeeks 2025a.) The Thread network uses the newest IP (Internet Protocol) addressing standard, IPv6. The IPv6 standard provides device-to-device and device-to-cloud connections, and it allows already existing IP based application layers to work on top of Thread without proprietary gateways of translators. (Nordic Semiconductor 2020.)

UDP

The User Datagram Protocol (UDP) is one of the core protocols of the Internet Protocol. (GeeksforGeeks 2025b.) UDP is a transport-layer communication protocol that is used for messaging between devices for mesh establishment and maintenance. Thread Networks also supports the TCP (Transmission Control Protocol) or any other IPv6-based protocols for application layer communication. (Thread Group 2020.)

Security

Thread provides security at the network layer. All the network communication is encrypted and authenticated using the 802.15.4. security mechanisms. Devices are not allowed to join the network if they are not authenticated. (Nordic Semiconductor 2020.)

Application Layer

Any low bandwidth application layer that can run over IPv6 can run over Thread. Thread can support multiple applications layers simultaneously. (Thread Group 2020.)

2.2 Thread network

The Thread network consists of two kinds of device types and two kinds of forwarding roles. Different Thread devices are combinations of these two features.

2.2.1 Forwarding roles of Thread devices

The Thread devices have two forwarding roles, as is shown in Figure 2. A device can either be a Router or an End Device.

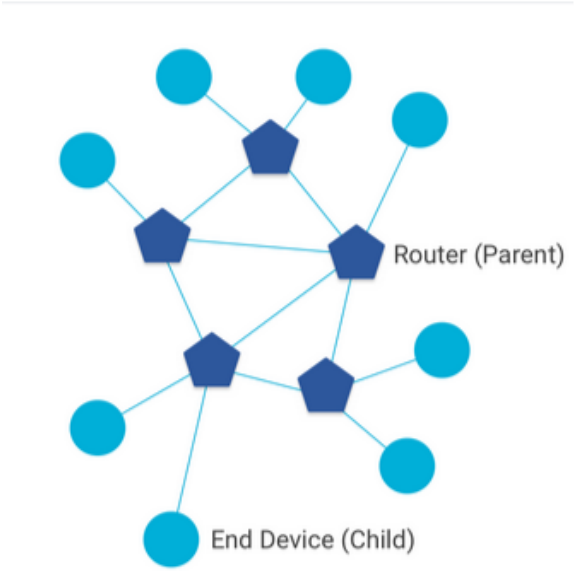

FIGURE 2. Thread Forwarding Roles (OpenThread 2025.)

Routers

Routers are parent nodes that forward packets for devices in the Thread network. The routers always keep their transceiver on. A Thread network can have up to 32 Routers. (OpenThread 2025.)

End Devices

End devices are child nodes that communicate only with their Parent Router and do not forward packages to other devices in the network. End devices can disable their transceiver for lower power consumption. A Thread network can have up to 511 End Devices per Router. (OpenThread 2025.)

2.2.2 Thread device types

Thread devices are divided in two types that are Full Thread Devices and Minimal Thread devices. Figure 3 shows what Thread devices belong to each type.

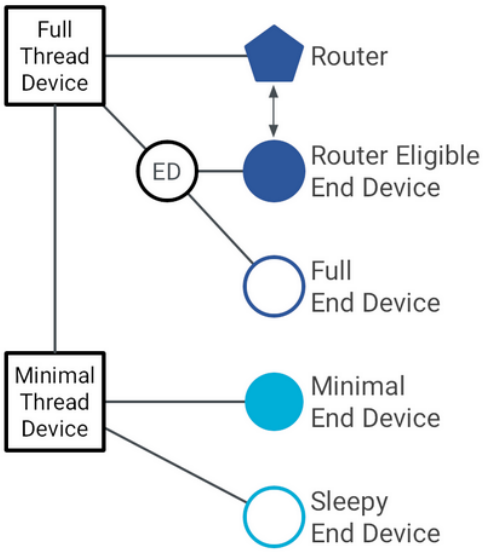

FIGURE 3. Thread Device Types (OpenThread 2025.)

Full Thread devices

Full Thread Devices have their radio on at all times. Their role is to subscribe to the all-routers multicast address and maintain IPv6 address mapping. Full Thread Device is the most versatile role in the Thread network. It can operate as a Router (parent) or as an End Device (child). (OpenThread 2025.)

Minimal Thread devices

Minimal Thread Devices can either always have their radio on or it can be turned off at times, depending on their forwarding role. Minimal Thread Devices do not subscribe to the all-routers multicast address, and they have the lowest requirements on memory size and power consumption. Minimal Thread devices are always End Devices (child nodes). (OpenThread 2025.)

2.2.3 Thread device types and different forwarding roles

When the two device types and two forwarding roles are combined, it generates different kinds of Thread devices which each have their own tasks. A Thread mesh network with all the different types of devices connected could look like the network illustrated in Figure 4.

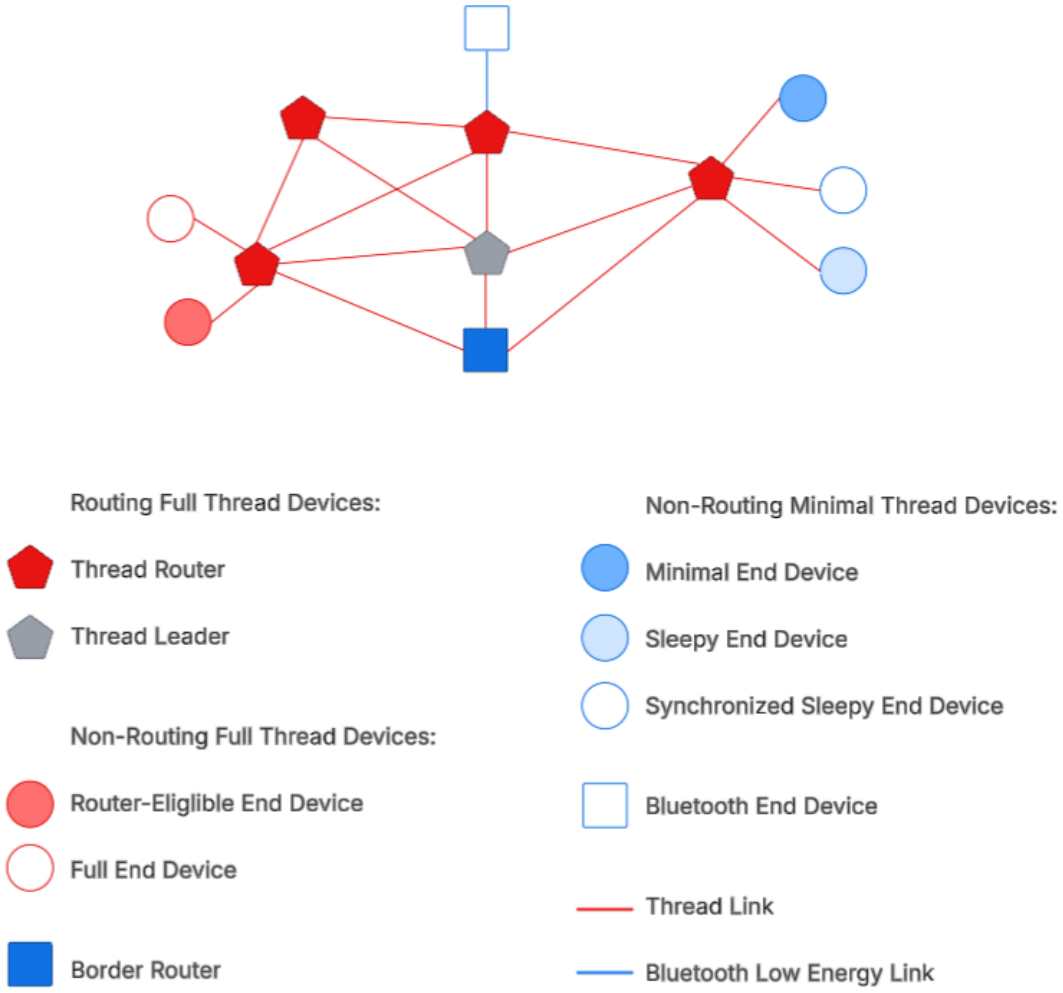

FIGURE 4. Thread Device Types and Forwarding roles

Routing Full Thread Devices

Thread Routers and Thread leaders are Routing Full Thread Devices. Thread Routers provide routing services for the Thread devices. Their role is to provide joining and security. All Thread routers can be downgraded to Router-Eligible End Devices or selected to be Leaders. Thread Leader is a role that only one of the Thread Routers is elected for. It manages the other Routers in the Thread network. If the Leader does not work for some reason, another Router will be promoted to be the Leader. (Thread Group 2020.)

Non-Routing Full Thread Devices

Router Eligible End Devices (REED) are End Devices that have the capability to be upgraded as a Router. Full End Devices (FED) are End Devices that are not able to become Routers or Leaders, but they can act as a Boarder Router. Non-Routing Full Thread Devices are child nodes. (Thread Group 2020.)

Non-Routing Minimal Thread devices

There are four kinds of Non-Routing Minimal Thread devices: Minimal End Devices (MED), Sleepy End Devices (SED), Synchronized Sleepy End devices (SSED) and Bluetooth End Devices (BED). Minimal End Devices have their radios on all the time and communicate only through their Parent Router. Sleepy End Devices are disabled most of the time but wake up occasionally to poll messages from their parent router. Synchronized Sleepy End devices have their radio turned off during idle periods. They wake up periodically at scheduled intervals to listen to messages from their Parent Router. Bluetooth End Devices communicate over a Bluetooth Low Energy link, and they communicate only with their Parent Router, which is a Bluetooth LE Bridge Router. (Thread Group 2020.)

Border Router

The Thread Boar Router is a special role of a Full Thread Device where the device provides a connection for sending data between the Thread network and non-Thread networks as Wi-Fi and Ethernet. A Thread network can have several Border Routers. (Thread Group 2020.)

2.3 Thread compared to other systems

Different smart home systems have their advantages and disadvantages. Five most common systems are compared in the Table 1 and a with a brief introduction of each system below.

TABLE 1. Smart home systems compared

	Thread	BLE	Wi-Fi	Z-wave	Zigbee
Range	30 m	100 m	20 m (indoors)	30-100 m	10-100 m
Power efficiency	High	Very high	Low	High	High
Hub Required	No	No	No	Yes	Yes
Data Rate	250 Kbps	2 Mbps	9.6 Gbps	Up to 100 Kbps	250 Kbps
Devices per network	16384	20-50	30-250	4000	(65000) 240
Frequency Band	2.4 GHz	2.4 GHz	2.4GHz, 5GHz, 6GHz	869-908 MHz	2.4 GHz
Mesh Support	yes	optional	no	yes	yes

Wi-Fi

Wi-Fi is a wireless communication technology that is used for connecting devices to the internet or to other devices. It uses radio waves for data transmission. Devices are typically connected to a router, but they can also be connected to a smart phone or a computer. (HowToGeek 2023.) Wi-Fi is the most used technology for smart home connections. The advantages of using Wi-fi compared to Thread are that it is a widely supported technology so there are a lot of devices available, and it has high bandwidth, which makes it more suitable for video and music streaming and security cameras. Compared to Thread, Wi-Fi has a low range and high-power consumption that makes it unsuitable for low-power smart

home systems. A smaller number of devices can be connected in the network compared to Thread networks. (Dlachenko 2023.)

Bluetooth Low Energy

Bluetooth is a wireless communication technology that was designed for transmitting data between devices within a short range. BLE (Bluetooth Low Energy) is the power-optimized version of Bluetooth. It was designed specifically for low power devices with small data transmission. (Mohammad Afanef 2022.) The advantages of BLE compared to Thread are that it is supported by a larger number of devices and it can be used with most smartphones. BLE uses even less power than Thread and this is why it is optimal for smart locks and wearables. The Thread network can be extended for a larger range through its mesh network, and it can have more devices connected to its network than BLE. (Dlachenko 2023.)

Zigbee

Zigbee is a low-power wireless network suitable for smart home solutions. It supports star, tree and mesh topologies. (Walsh 2025.) In this protocol each device acts as a node which makes the network more reliable. The mesh also increases the coverage of the network. (Charlton 2021.) The Zigbee network has a hub that controls the devices in the network and communicates with other networks and systems. Zigbee has a lot of similarities compared to Thread. The main difference in the systems is that Thread is an IPv6-based protocol that works with various application layers whereas Zigbee is a complete IoT framework that includes both its own application and network layers. Zigbee's advantage is that it has more compatible devices in the market than Thread, and it can support more devices in its network. (Walsh 2025.)

Z-wave

Z-wave is a low-power communication protocol that operates in the sub-GHz frequency band. (Z-wave Alliance 2025.) The devices form a mesh network and use a hub to connect to the internet. The devices in the network act as repeaters, which extends the coverage of the network. Thread and Z-wave differ in their application layer. Z-wave has a defined application layer built in the protocol stack and Thread works with various application layers. The advantage of Z-wave compared to Thread is its sub-GHz frequency band, for which reason the devices connected to the network are not as susceptible to interference from other devices. The disadvantage of Z-wave is that the network is not able to support as many devices as Thread can and the data rate is smaller than in the Thread networks. (RFWirelessWorld 2025.)

Conclusion

When selecting a communication technology for the project, Thread stands out as an ideal protocol due to its energy efficiency, scalability, and support for mesh networking. It offers interesting features to study, and it is well-suited for developing a low-power sensor platform. Its IPv6-based design also simplifies internet connectivity and enables seamless device-to-device communication, making cloud integration more straightforward.

3 SYSTEM AND COMPONENT REQUIREMENTS

3.1 System requirements

The system requirements define the high-level goals and capabilities for the complete end-to-end system, combining all major components: the sensor platform, Thread Border Router, and web application. The following requirements in Table 2 define the essential and optional functionalities that ensure the system performs reliably but also remains flexible for possible expansions in the future.

The overall system architecture is illustrated in Figure 5. Sensor platforms form a mesh network and communicate through a Thread Border Router, which acts as a gateway forwarding data to the cloud service. From there, the data is accessible to the user via the web application. This architecture supports scalable systems where integration of multiple sensor nodes is possible.

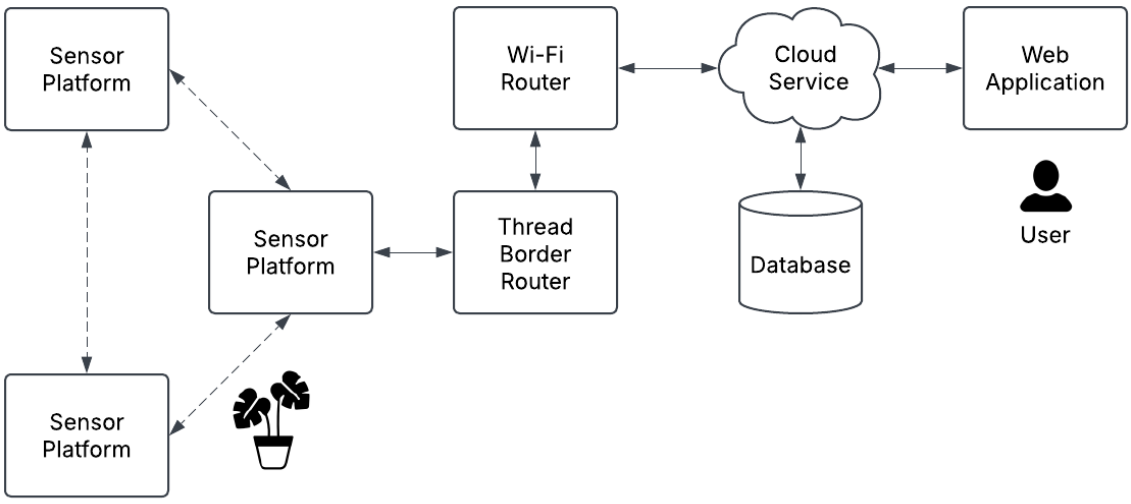

FIGURE 5. Block diagram of the system. Sensor platforms form a mesh network and connect to the internet through a Thread Border Router.

TABLE 2. Overall system requirements

ID	Type	Description
SYS-REQ-1	Required	The overall system must enable end-to-end data transmission between the sensor platform and the web application.
SYS-REQ-2	Required	The overall system must be suitable for use in various types of IoT projects.
SYS-OPT-1	Optional	The overall system must allow configuration without the need for deep technical knowledge.
SYS-OPT-2	Optional	The overall system must be easily maintainable and updatable.

These essential requirements were chosen to ensure the system provides meaningful value to the user. SYS-REQ-1 focuses on the main goal of delivering sensor data to user through the internet, while SYS-REQ-2 ensures that the platform is general enough to be applied across different types of IoT projects.

3.2 Sensor platform requirements

The sensor platform is the first component in the whole system and is responsible for collecting sensor data and transmitting it forward. The following requirements in Table 3 define the essential and optional functionalities that ensure the platform can support a variety of IoT applications.

TABLE 3. Sensor platform requirements

ID	Type	Description
SP-REQ-1	Required	The sensor platform must communicate wirelessly using the Thread protocol.
SP-REQ-2	Required	The sensor platform must support collection of data from variety of sensor types.
SP-REQ-3	Required	The sensor platform must provide interfaces for connecting external sensors and devices via I2C, analog input pins and digital I/O pins.
SP-REQ-4	Required	The sensor platform must be able to revive from sudden power loss and reboot on its own and be able to join the Thread network after that.
SP-OPT-1	Optional	The sensor platform must support two-way communication with the web application for sending sensor data and receiving commands.
SP-OPT-2	Optional	The sensor platform must maintain a stable wireless network connection within a range of 10 meters in an indoor environment.
SP-OPT-3	Optional	The sensor platform must be able to work with common Thread Border Routers.

These essential requirements were designed to ensure that the sensor platforms can work independently and form a reliable mesh network that is able to heal in cases where one node fails. Optional requirements expand the usefulness of the sensor platform.

3.3 Thread Border Router requirements

The Thread Border Router acts as a gateway between the local mesh network of sensor platforms and external IP-based networks, such as Wi-Fi. The following requirements in Table 4 define the essential capabilities and additional features needed to maintain stable communication and enable seamless integration between the sensor platforms and cloud service.

TABLE 4. Thread Border Router requirements

ID	Type	Description
TBR-REQ-1	Required	The TBR must provide a reliable connection between the sensor platform and the cloud service.
TBR-REQ-2	Required	The TBR must automatically recover from power loss and support reboot.
TBR-OPT-1	Optional	The TBR must support the addition of new sensor platforms to the Thread network with minimal user effort.

These requirements were designed to make sure the TBR will perform reliably in its critical role in bridging the local mesh network and external services.

3.4 Web application requirements

The web application serves as the user-facing component of the system, enabling the user to view and analyse the sensor data. To support these functionalities, the application must be responsive, functional and efficient in retrieving and presenting real-time information. The following requirements in Table 5 define the essential and additional functionalities of the web application.

TABLE 5. Web application requirements

ID	Type	Description
WA-REQ-1	Required	The web application must fetch sensor data from the database.
WA-REQ-2	Required	The web application must show real-time and historical sensor data in graph format.
WA-REQ-3	Required	The web application must allow users to define thresholds that trigger alerts.
WA-OPT-1	Optional	The web application must allow users to configure sensors from the UI.
WA-OPT-2	Optional	The web application must have responsive layouts that scales for smartphones, tablets and laptop screens.

These requirements were designed to support the core goals of the web application: enabling access to sensor data, supporting real-time monitoring, and maintaining usability across different device types.

4 SENSOR PLATFORM

4.1 Analysis

A major part of the project was designing and building a sensor platform, where optional sensors could be attached and the features of the Thread network could be examined and tested practically. The data was then forwarded to a Cloud service as shown in Figure 6.

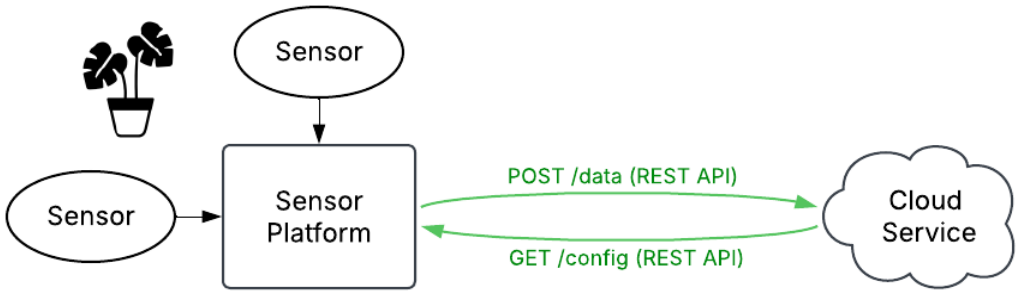

FIGURE 6. Sensor Platform

With the time and resources in use, it was impossible to build nearly as big of a mesh network that could be built if the whole potential of Thread would be put in use, but in the project some key components were used to display the different Thread device types and forwarding roles.

There are several devices in the market that can be used to build a Thread network and at first the suitable devices had to be chosen. The devices were selected based on the requirements listed in chapter 3, with particular emphasis on the following criteria:

- SP-REQ-1: The system should communicate wirelessly using the Thread protocol.

- SP-REQ-3: The sensor platform must provide interfaces for connecting external sensors and devices via I2C, analog input pins and digital I/O pins.
- SP-OPT-2: The platform should maintain stable wireless network connection within a range of 10 meters in an indoor environment.
- SP-OPT-3: The sensor platform should be able to work with common Thread Board Routers.

According to these requirements, two ESP32-H2-DevKitM-1 boards were chosen. They were used for setting up a small Thread mesh network and for sending sensor data wirelessly to a Thread Board Router, which is introduced in more detail in chapter 5.

4.2 Design

The design of the sensor platform began with choosing the suitable devices, installing the required software and setting up the Thread network.

4.2.1 Hardware

The setup of the ESP32-H2-DevKitM-1 requires a USB-A to USB-C cable and a computer that runs Windows, Linux or macOS. (Espressif Systems 2025a.)

ESP32-H2-DevKitM-1

The ESP32-H2 development board shown in Figure 7 is an entry-level development board that is produced by Espressif. The development board is ideal for systems that require low power and secure short-range wireless communication. It has built-in features that enable Thread, and it meets the requirements listed above, which makes it a suitable part of the sensor platform system. (Espressif Systems 2025a.)

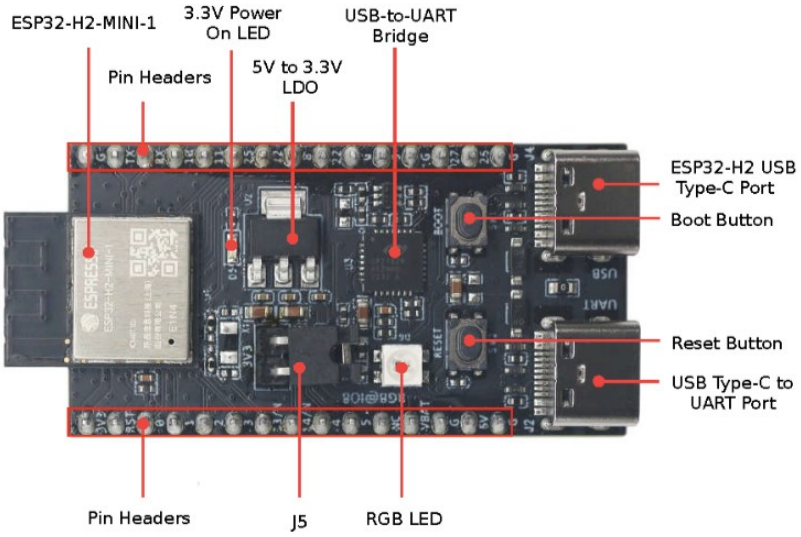

FIGURE 7. ESP-H2-DevKit-1 (Espressif Systems 2025a.)

Key components of the development board:

- **ESP32-H2MINI-1:**
  A powerful and compact Bluetooth Low Energy and IEEE 802.15.4. combo module that supports a wide range of peripherals (UART, SPI, I2C, ADC, USB). The module comes with a PCB antenna. (Espressif Systems 2025c.)
- **Pin headers:**
  The board can be connected to peripherals with jumper wires, or it can be mounted on a breadboard for connections. (Espressif Systems 2025b.)
- **ESP32-H2 USB Type-C port:**
  The USB Type-C port on the ESP32-H2 chip compliant with USB 2.0 full speed. It is capable of up to 12 Mbps transfer speed. Primary port for communication and flashing firmware on the board. (Espressif Systems 2025b.)
- **USB type-C to UART Port:**
  A power supply for the board as well as a communication interface between a computer and the ESP32-H2 chip via USB-to-UART bridge. (Espressif Systems 2025b.)

4.2.2 Software

When using the ESP32-H2 development board, following software must be installed:

- Toolchain to compile code for ESP32.
- Build tools: CMake and Ninja to build a full Application for ESP32.
- ESP-IDF (Espressif IoT Development Framework) that essentially contains an API (software libraries and source code) for ESP32-chips and scripts to operate the Toolchain.

The software can be installed through an IDE (Eclipse Plugin or VSCode Extensions), or manually with specific instructions according to the operating system used. The instructions can be found on the Espressif-website. (Espressif Systems 2025c.)

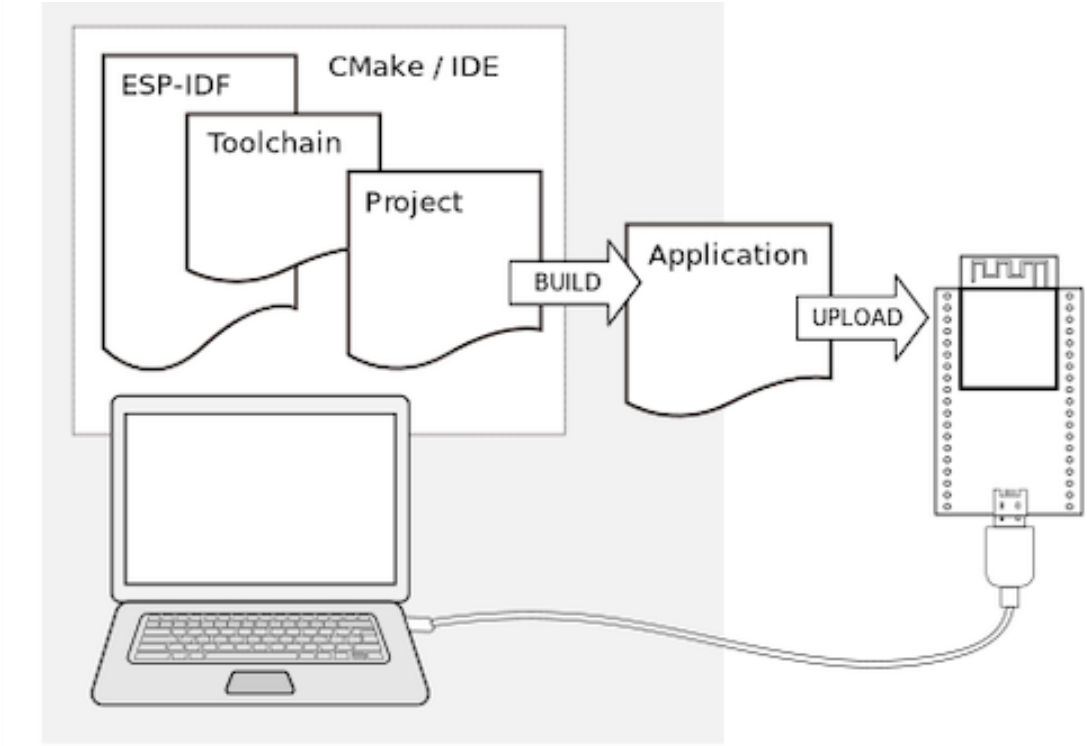

FIGURE 8. ESP-IDF (Espressif Systems 2025c.)

The software for the ESP32-H2 development boards was installed on a computer with a Windows operating system. The ESP-IDF was installed in few steps that were not directly according to the instructions, but were proved to be effective:

- To get the recommended ESP-IDF version for forming a Thread network, the ESP-IDF version 5.2.4. was cloned from a GitHub repository. (Espressif Systems 2025f.)
- Next, the Vs-code extension was configured and the option “find ESP-IDF in your system” was selected.
- At last, the Eclipse Plugin was installed with minimal extensions, in addition of the existing software. This was done to get the Eclipse ESP-IDF terminal in use, because some of the commands did not work in the Windows Command Prompt and the Eclipse plugin terminal window seemed easier to use than the terminal window in the Vs-Code extension.

4.2.3 Forming the Thread network

Espressif provides instructions for building the images for the Thread Board Router and Client devices and for forming the Thread network with them. (Espressif Systems 2025d.) In the project, an ESP Thread Board Router was set up according to the instructions and the client example was built and flashed on two ESP32-H2 development boards. Espressif provides a website for monitoring the formed Thread network in real time. The topology updated when a device joined, dropped out or changed its role. As shown in Figure 9, The Thread Board Router has taken the role of a Thread Leader, one of the ESP32-H2 boards is a Router and the other ESP32-H2 board is a Child device.

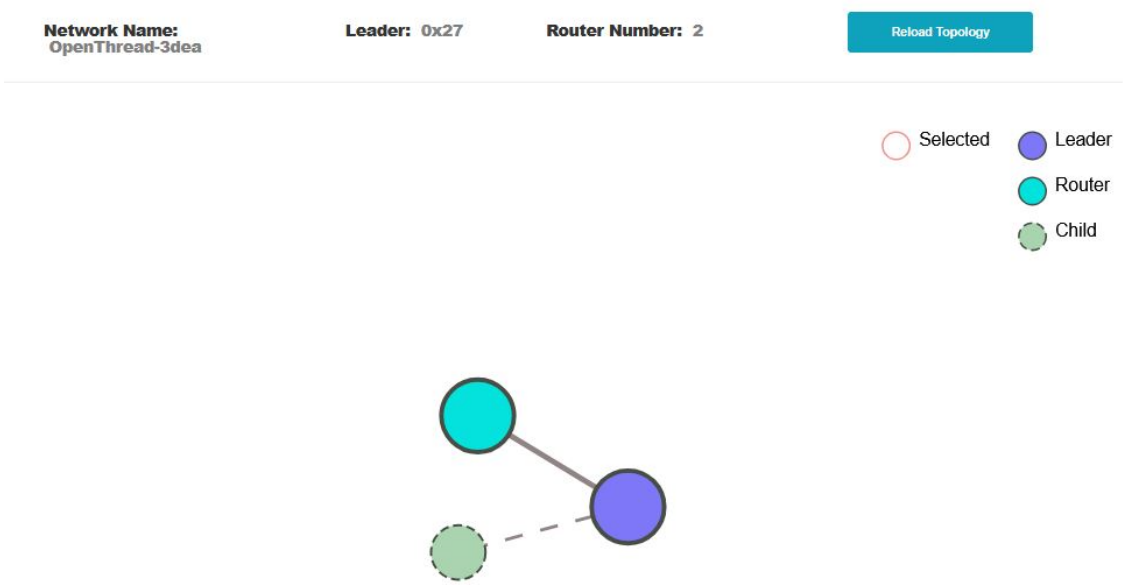

FIGURE 9. Thread Network Topology 1

In the Figure 10, the second ESP32-H2 board has changed its role from a Child to a Router. The devices took the roles automatically, but if customized settings are wanted, the roles can also be configured manually for each device in menuconfig. (Espressif Systems 2025d.)

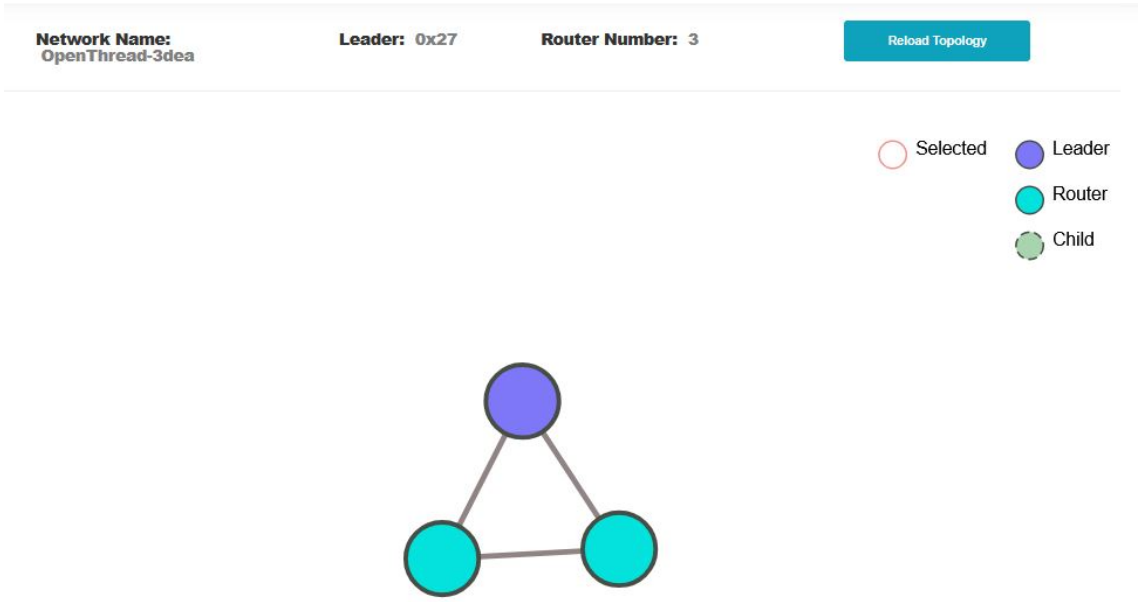

FIGURE 10. Thread Network Topology 2

4.3 Implementation

When the devices are properly set up and the required software is successfully built and flashed on the Thread Board Router and on the ESP32-H2 boards, they form a small Thread mesh network. One of the ESP32-boards takes the role of a Thread Leader and connects to the Thread Board Router. The other ESP32-H2 device can be set up in different Thread device roles, depending on what features are wanted. The Figure 11 shows different ways how the Thread network can be formed with the devices in use.

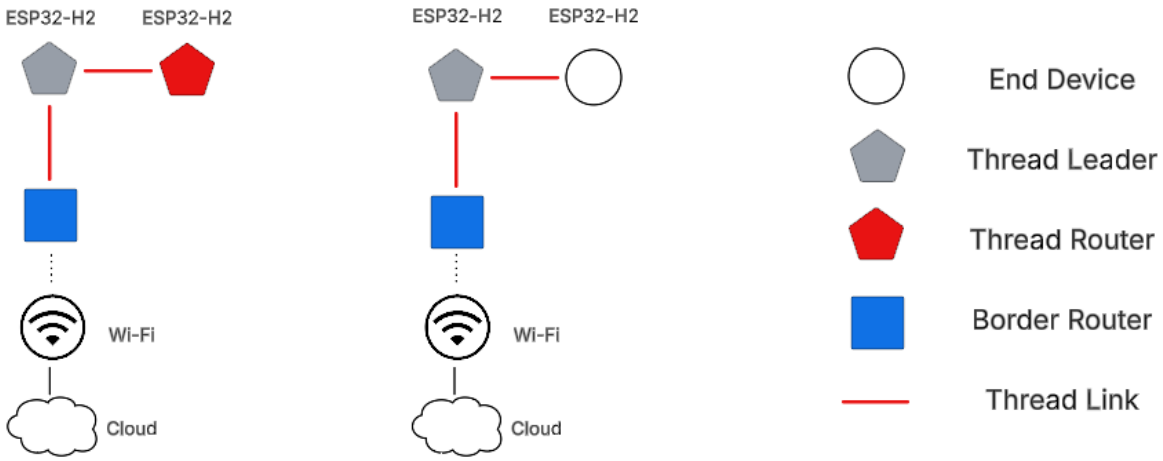

FIGURE 11. Alternative formations of a Thread mesh network

This Thread network allows connecting various peripherals to it and gives many options for building different kinds of sensor platform systems. Even with only two ESP32-H2 boards in use, several peripheral devices can be connected to the sensor platform.

5 THREAD BORDER ROUTER

5.1 Analysis

A Thread Border Router (TBR) is a device, usually a smart speaker, streaming device or dedicated smart home router, that acts as an intermediary between Thread-enabled IoT devices and other IP-based networks, such as Wi-Fi or Ethernet. This allows Thread-enabled devices to be controlled over the internet, even when the user is not at home. In this project, a dedicated TBR is used to connect the sensor platforms to the internet.

In this project, a dedicated TBR is used to connect the sensor platforms to the internet like shown in Figure 12. Alternative approaches included using a third-party Thread Border Router or a Raspberry Pi with OpenThread, which is an open-source implementation of Thread. With third-party party TBR's, it would be possible to send sensor data to mobile app, but sending data to cloud turned out to be very difficult to achieve. Since cloud integration was selected as a project feature, a TBR that supports internet connectivity was required. The best options were the Raspberry Pi and Espressif's TBR, and ultimately, Espressif's ESP Thread Border Router was chosen.

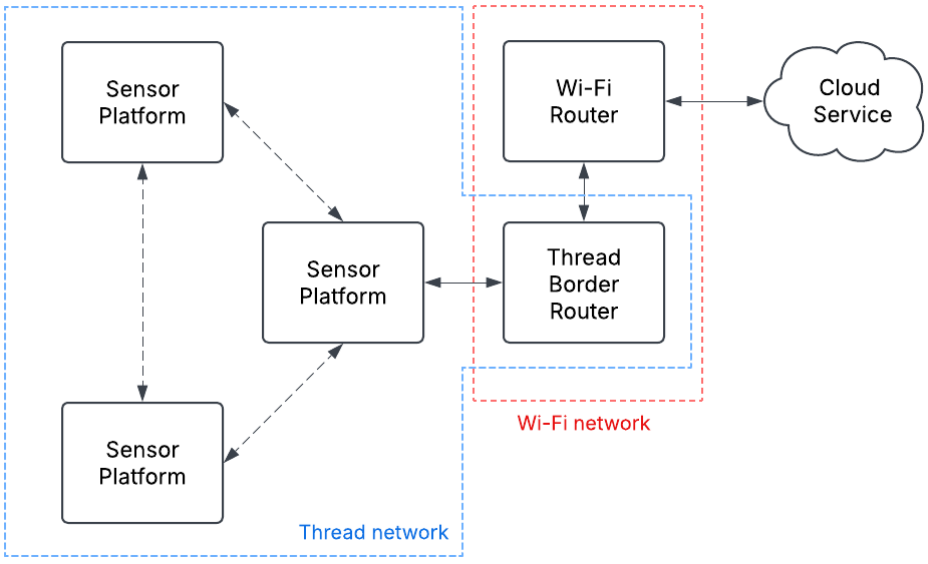

FIGURE 12. TBR bridges the Thread network with the Wi-Fi network

5.2 Design

Thread Border Routers from major companies like Apple and Google were not suitable for this project, as building a system that sends sensor data to the cloud and receives commands in return would have been difficult to develop using their closed ecosystems. This left us with two viable options: using a Raspberry Pi with an external Thread antenna, or the ESP Border Router by Espressif. ESP Border Router was finally selected as it is more affordable and better suited for this project. Espressif also provides comprehensive documentation for setting up the TBR, along with useful example code for testing.

Device overview

The ESP Thread Border Router in Figure 13 utilizes two System-on-Chips (SoCs): an ESP32-S3 and an ESP32-H2. The ESP32-S3 acts as the host SoC, providing connectivity via Wi-Fi, while the ESP32-H2 functions as a radio co-processor (RCP), enabling access to the Thread protocol. (Espressif Systems 2025e.)

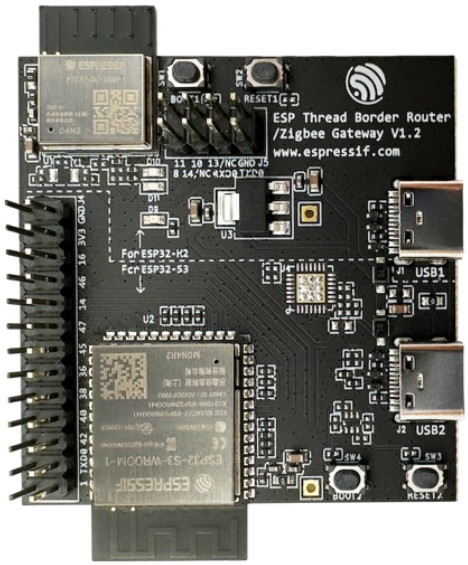

FIGURE 13. ESP Thread Border Router (Espressif Systems 2025e.)

5.3 Implementation

To set up the Thread Border Router, Espressif's official example project *basic_thread_border_router* was used and flashed using the ESP-IDF extension for Visual Studio Code. ESP-IDF is Espressif's development framework for their SoC's and provides access to build tools, flashing utilities and serial monitoring. This allowed easy configuration, debugging and testing during development.

ESP-IDF

Espressif's development framework offered easy to use solution for configuring both the TBR and ESP32-H2 used in the sensor platform. Figure 14 shows the graphical user interface provided by the ESP-IDF for configuring the TBR's settings.

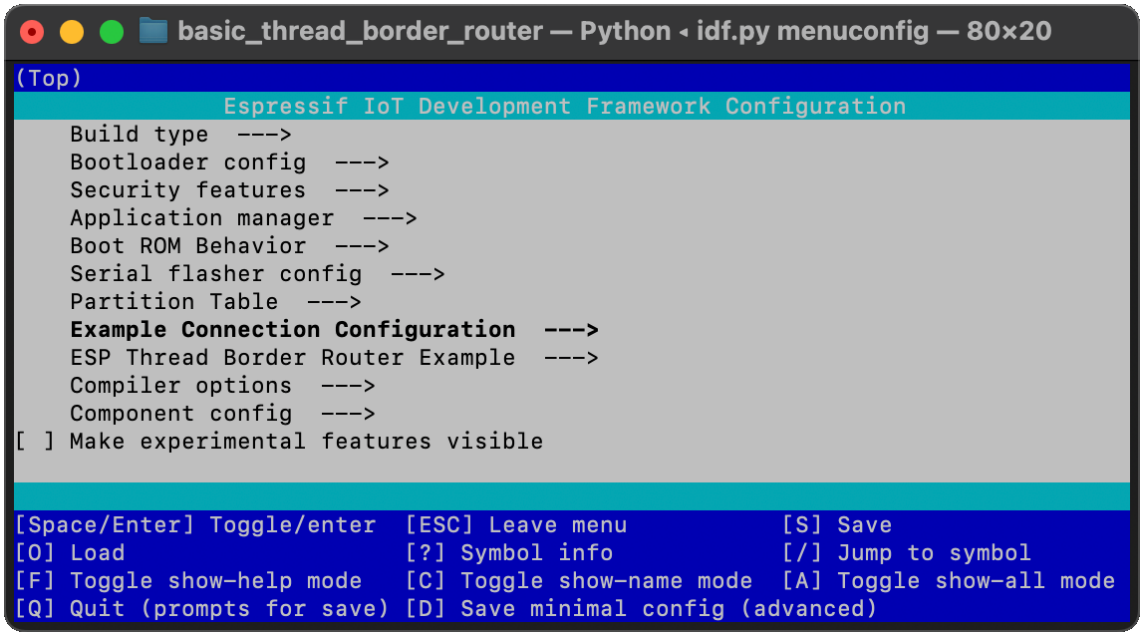

FIGURE 14. ESP-IDF menuconfig window

6 WEB APPLICATION

6.1 Analysis

The web application is the main way for the user to interact with the overall system, and for that reason, the focus is on making it responsive and intuitive to use. The application displays sensor data and provides interface for updating the sensor settings. An overview of the web application's architecture and data flow is shown in Figure 15.

To enable cloud integration for the system, a cloud hosting service was selected to host both the backend and frontend. If access to the data had only been required locally, a self-hosted server on a Raspberry Pi microcomputer would have been sufficient.

For the cloud hosting, several alternatives were considered, but as there was already some knowledge about using Cloudflare's services, that was chosen to host the systems database, backend and frontend code. Cloudflare provides free of charge place to host server-side code in form of Workers, that is a serverless code execution service.

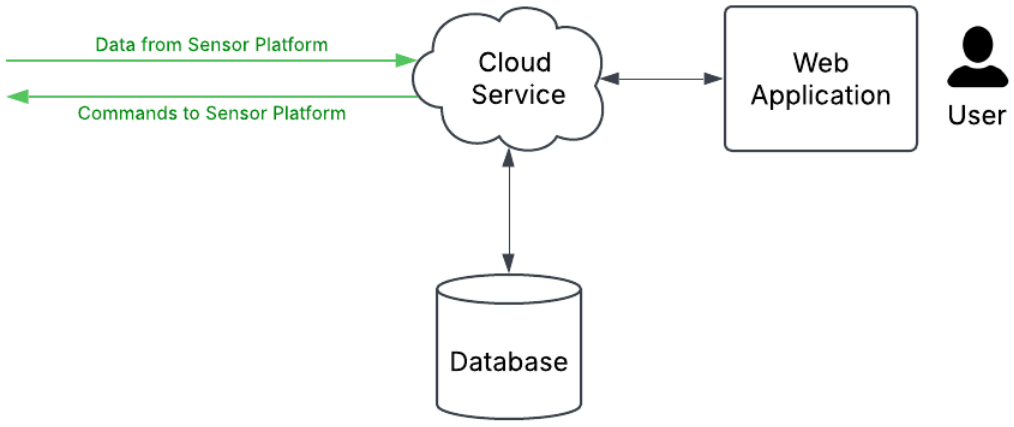

FIGURE 15. Block diagram showcasing relations of the different components of the web applications and the two-way communication between the sensor platform and web application.

6.2 Design

The design of the web application focused on creating a clean and responsive interface to present sensor data. The web application consists of three main parts: the frontend interface, the backend API layer, and the data layer in the Cloudflare D1 SQL database. The backend communicates with the frontend and sensor platforms through REST API endpoints managed by Cloudflare Workers.

6.2.1 Frontend and UI design

For efficient frontend development, various frameworks were considered. Given the project's simple requirements and existing knowledge of web frameworks, SvelteKit was selected. It offers a relatively small bundle size and has simple syntax (Czerniuk 2025).

The frontend was designed with a mobile-first approach, meaning the website was first optimized for mobile devices and then scaled up for larger screen sizes. This mobile-first approach is illustrated in Figure 16. The focus of the UI design was on presenting the most relevant information immediately and allowing users to access more detailed data with a single click.

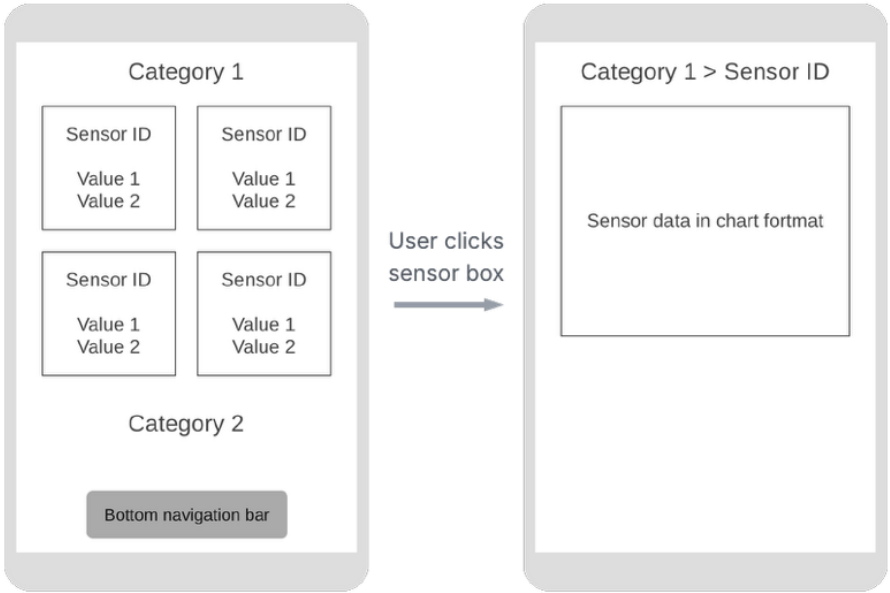

FIGURE 16. Wireframe of the web application

6.2.2 Backend architecture

Database

The database illustrated in Figure 17 consists of five tables:

- **sensors**: Each sensor has unique identifier (*sensor_id*) in the form of UUID v4, ensuring uniqueness across all sensor platforms. This also serves as the primary key in this table. Each sensor has a *category_id* that references to categories table and links the sensor to user defined category (e.g. terrace, living room, etc.).
- **categories**: Stores category names that users can assign to sensor platforms for grouping in the web application.
- **sensor_data**: Stores data readings for each sensor platform. Each entry contains a *timestamp* and a *values* field that holds a JSON object with one or more measurements.
- **sensor_config**: Contains configuration details for each sensor platform.
- **sensor_thresholds**: Stores alert thresholds for specific sensor values. Each threshold entry is linked to a sensor by *sensor_id* and defines a key (e.g. “temperature”) with min_value and/or max_value fields. These thresholds are used by the system to detect when sensor’s values exceed user defined values and trigger an alert.

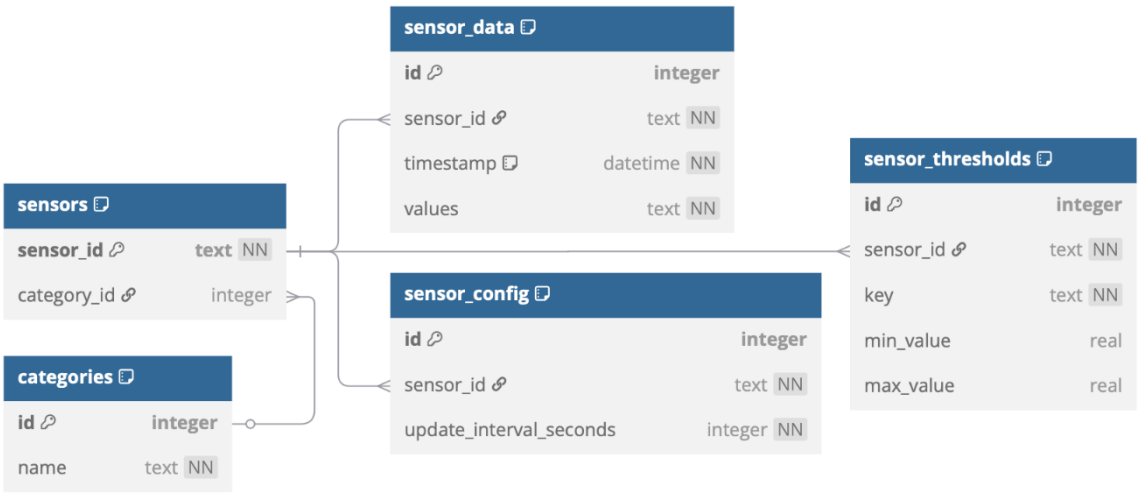

FIGURE 17. Database schema

REST API

A Representational State Transfer (REST) Application Programming Interface (API) is used to bridge web portal's backend and the sensor platforms. Each sensor platform independently communicates with the REST API's endpoints to send data to and receive commands from the web application. Figure 18 illustrates the data flow between different core components using the REST API.

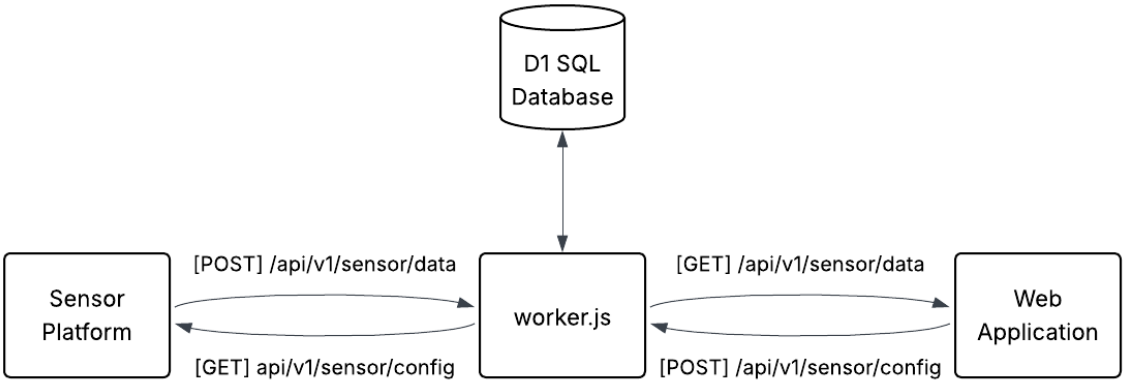

FIGURE 18. Data flow between components using the REST API

Security and authentication

By default, the API is accessible to anyone who knows its URL and endpoint. Although no sensitive information processed, basic authentication was implemented to prevent unauthorized access to the database. This was achieved using Authorization field in the header of each REST API call. The field contains a static secret key, which is checked in the worker.js code every time a request is made. If the field is missing or the key does not match the one defined in the worker code, the request is rejected.

6.3 Implementation

The web application shown in Figure 19 was built using SvelteKit and all the frontend and backend code was hosted on Cloudflare's platform, taking advantage of their Workers for server-side logic and D1 SQL as the backend database.

Cloudflare Workers were used to handle the REST API calls that were used for communication between the sensor platforms and the web application. Each sensor platform sends sensor values to the database making HTTP requests to specific API endpoint managed by the Workers. The Workers validates the requests, extracts the sensor data and insert it into the D1 SQL database.

In addition to handling incoming data from the sensor platforms, the Workers also serve as the backend for the frontend, responding to API calls that retrieve the latest sensor readings for display in the web application.

FIGURE 19. Completed UI for the web application

7 SYSTEM TESTING, INTEGRATION AND PERFORMANCE

7.1 Component-level testing

Component-level testing is conducted to verify all individual units of the project work as expected on their own. Whenever possible, core components are tested first in isolation to ensure they meet their essential requirements and behave correctly. However, in cases where components are inherently interconnected, such as the sensor platform forming a Thread network with the TBR, testing is performed in a partially integrated manner before moving to full integration testing.

7.1.1 Sensor platform testing

This section presents the tests conducted on the sensor platform to ensure its core functions operate reliably. The goal of this testing phase is to verify the platform's ability to boot correctly, initialize its components, acquire and process sensor data.

In this testing phase ESP-IDF's monitoring feature was used to detect possible errors and to observe the state of the Thread network. Table 6 documents the test cases and results for the sensor platform.

TABLE 6. Sensor platform test cases and results

SP-1	Verify wireless communication using Thread protocol
Objective	Confirm that the sensor platform communicates wirelessly using the Thread protocol.
Related Requirement	SP-REQ-1
Method	Set up the sensor platform can verify from ESP-IDF's monitor that it connects to the Thread network.

Expected	Device successfully join a Thread network and maintains the connection.
Result	Passed
Notes	-
SP-2	Verify data collection from various sensor types with different connection interfaces
Objective	Ensure that the sensor platform can collect and process data from multiple types of sensors (e.g. temperature, humidity, soil moisture, motion) with various of connection interfaces.
Related Requirement	SP-REQ-2, SP-REQ-3
Method	Connect different types of sensors to the platform and verify data collection via terminal window. Use I2C connection, analog pins and digital I/O pins for connecting the sensors.
Expected	All connected sensor types send valid data
Result	Passed
Notes	Connections worked but there were problems with soil moisture sensor and its readings.
SP-3	Verify recovery after sudden power loss
Objective	Confirm that the sensor platform can recover from sudden power loss and rejoin the Thread network automatically without user intervention.
Related Requirement	SP-REQ-4
Method	Remove power source from the sensor platform and reconnect it. Check if it rejoins the Thread network without user intervention. Verify connection from TBR's web portal it self-hosts.
Expected	Sensor platform resumes normal operation and reconnects.
Result	Passed
Notes	-

7.1.2 Thread Border Router testing

This section presents the tests conducted on the Thread Border Router (TBR), which plays central role in bridging the sensor platforms with external networks. The goal of this testing phase is to verify that the TBR initialized correctly, maintains stable communication between networks and successfully forwards sensor data to the cloud.

In this testing phase ESP-IDF's monitoring feature was used to detect possible errors and to observe the state of the Thread network. Table 7 documents the test cases and results for Thread Border Router.

TABLE 7. Thread Border Router test cases and results

TBR-1	Verify reliable connection between sensor platform and cloud service
Objective	Ensure that the TBR maintains a reliable and stable connection between the sensor platform and the cloud service.
Related Requirement	TBR-REQ-1
Method	Set up the TBR and connect a sensor platform to it. Run data transmissions continuously for an extended period and monitor disconnections or transmission errors.
Expected	No connection drops or data transmission failures.
Result	Passed
Notes	-
TBR-2	Verify automatic recovery after power loss and reboot
Objective	Ensure that the TBR can recover automatically from a power loss or reboot without manual intervention.
Related Requirement	TBR-REQ-2
Method	Power off the TBR and then turn it back on. Observe if it reconnects to the Thread network and Wi-Fi network automatically.

Expected	The TBR resumes normal operation after reboot without user intervention.
Result	Passed
Notes	-
TBR-3	Verify easy addition of new sensor platforms
Objective	To ensure that new sensor platform can be added to the Thread network with minimal user effort.
Related Requirement	TBR-OPT-1
Method	Attempt to add a new sensor platform to the Thread network using a provided UI or configuration tools. Measure the number of steps required.
Expected	New sensor platform is successfully added with minimal user interaction.
Result	Failed
Notes	Setup takes many steps. More work is required to make it more user-friendly.

7.1.3 Web application testing

This section presents the tests conducted on the web application, which serves as the user interface for accessing and visualizing sensor data collected from the sensor platform. The focus is to verify that the web application functions correctly and meets all its component requirements.

Tests are conducted to confirm that the web application correctly retrieves and displays real-time sensor data from the backend, handles errors and operates reliably across different usage scenarios.

Postman was used simulate and verify API responses while browser developer tools and terminal logs were monitored during manual testing to detect errors. In Table 8 are documented the test cases and results for the web application.

TABLE 8. Web application test cases and results

WA-1	Verify successful data retrieval
Objective	Ensure that the web application correctly retrieves sensor data from the database.
Related Requirement	WA-REQ-1
Method	Trigger a sensor data retrieval action from the web application and monitor the database queries and results.
Expected	Sensor data appears correctly on the web application with no errors.
Result	Passed
Notes	-
WA-2	Verify display of real-time and historical data in graphs
Objective	Confirm that the web application displays real-time and historical sensor data visually in graph format.
Related Requirement	WA-REQ-2
Method	Collect live sensor data and retrieve older stored data. Check that both types are plotted correctly in the graphs.
Expected	Data points update live, and historical graphs load correctly with accurate timestamps.
Result	Passed
Notes	-
WA-3	Verify threshold alert functionality
Objective	Ensure that users can set threshold values that trigger alerts when exceeded.
Related Requirement	WA-REQ-3
Method	Define a threshold value and simulate sensor data that exceeds it. Observe if an alert is generated.
Expected	Alert is triggered when sensor value crosses the defined threshold.

Result	Failed
Notes	This functionality was not ready by the time testing was conducted.
WA-4	Verify sensor configuration from UI
Objective	Ensure that users can configure sensors directly via the web application user interface.
Related Requirement	WA-OPT-1
Method	Attempt to edit sensor configurations through the web UI.
Expected	Sensor configuration is successful without need of connecting to the sensor via USB-C.
Result	Failed
Notes	This functionality was not ready by the time testing was conducted.
WA-5	Verify responsive layout across devices
Objective	Confirm that the layout adapts to smartphones, tablets and laptops.
Related Requirement	WA-OPT-2
Method	Open the web application on multiple devices with different screen sizes and/or adjust browser screen sizes. Observe layout behaviour.
Expected	Web application layout remains functional and visually coherent on all tested devices.
Result	Passed
Notes	-

7.2 Full system integration testing

Full system integration testing is done to ensure that all individual components of the system, sensor platforms, TBR, and web application, work together. The objective of this testing phase is to validate the complete dataflow from the sensor platform to the web application via the TBR.

Table 9 documents the test cases and results for full system integration.

TABLE 9. Full system integration test cases and results

FS-1	Verify end-to-end data transmission
Objective	Ensure that sensor data is transmitted from the sensor platform to the web application.
Related Requirement	SYS-REQ-1
Method	Set up the sensor platform and send a data packet to the web application through REST API. Verify that it appears on the web application correctly.
Expected	The data is received correctly and without excessive delay.
Result	Passed
Notes	Only few sensor readings out of few hundred were not transmitted.
FS-2	Verify system operability in different IoT projects
Objective	Confirm that the system supports a range of IoT projects without needing core system changes.
Related Requirement	SYS-REQ-2
Method	Connect and test different sensor types with the system.
Expected	All sensors function properly with the system.
Result	Partially passed
Notes	Connection via I2C worked, but there were problems getting valid data from soil moisture sensor using analog pins for data transfer.

FS-3	Verify two-way communication between the sensor platform and the web application.
Objective	Ensure that the sensor platform can send sensor data to and receive commands from the web application.
Related Requirements	SP-OPT-2
Method	Send sensor data to the web application and issue a control command from the web application to the platform.
Expected	Data transmission and command response are successful.
Result	Partially passed
Notes	Data transfer from sensor platform to cloud works but to the other way transfer is not currently supported.
FS-4	Verify stable wireless communication within 10 meters indoors
Objective	Confirm that the sensor platforms maintain a stable connection within a 10-meter indoor range.
Related Requirement	SP-OPT-3
Method	Place a platform at various distances up to 10 meters and monitor network stability.
Expected	No connection drops occur within 10 meters indoor.
Result	Passed
Notes	The results exceeded expectations: the sensor platform was able to communicate with the TBR over 100 meters.
FS-5	Verify ease of configuration
Objective	Ensure that a user without advanced technical skills can configure the system.
Related Requirement	SYS-OPT-1
Method	Ask a non-technical user to add a new sensor to the network.
Expected	The user can configure the sensor successfully without assistance.

Result	Failed
Notes	System is not easy to setup without some technical knowledge, since configuration is done through command line interface.
FS-6	Verify maintainability and updatability of the system
Objective	Confirm that updates can be performed easily.
Related Requirement	SYS-OPT-2
Method	Perform a system update and a sensor replace. Check system behaviour after changes.
Expected	Updates and maintenance are completed without introducing errors and the full system works as intended.
Result	Failed
Notes	Firmware updates can be done but the process currently involves several steps and requires technical knowledge.

The full system integration testing confirmed that all essential system requirements were met. The sensor platform, Thread Border Router, and web application worked together as intended and formed functional end-to-end system. Data was reliably transmitted from the sensor platforms to the cloud via TBR, demonstrating successful system interoperability.

However, some optional requirements were not fully met. These included Over-the-Air updates of the sensor platform for easier deployment, alerts when sensor values exceed user defined thresholds,

Overall, the testing validated the system's core functionality and confirmed a solid foundation for further development.

7.3 Network behaviour testing

Thread protocol supports the creation of mesh network between the sensor nodes, making it interesting to test how the network forms when one node fails or when node roles change. With this testing setup the resilience of the mesh network was tested and observed.

7.3.1 Mesh network formation

With this testing phase the aim is to observe and test how Thread network is automatically formed when Thread-enabled devices are powered on and how nodes route traffic between each other. Initial testing setup is illustrated in Figure 20.

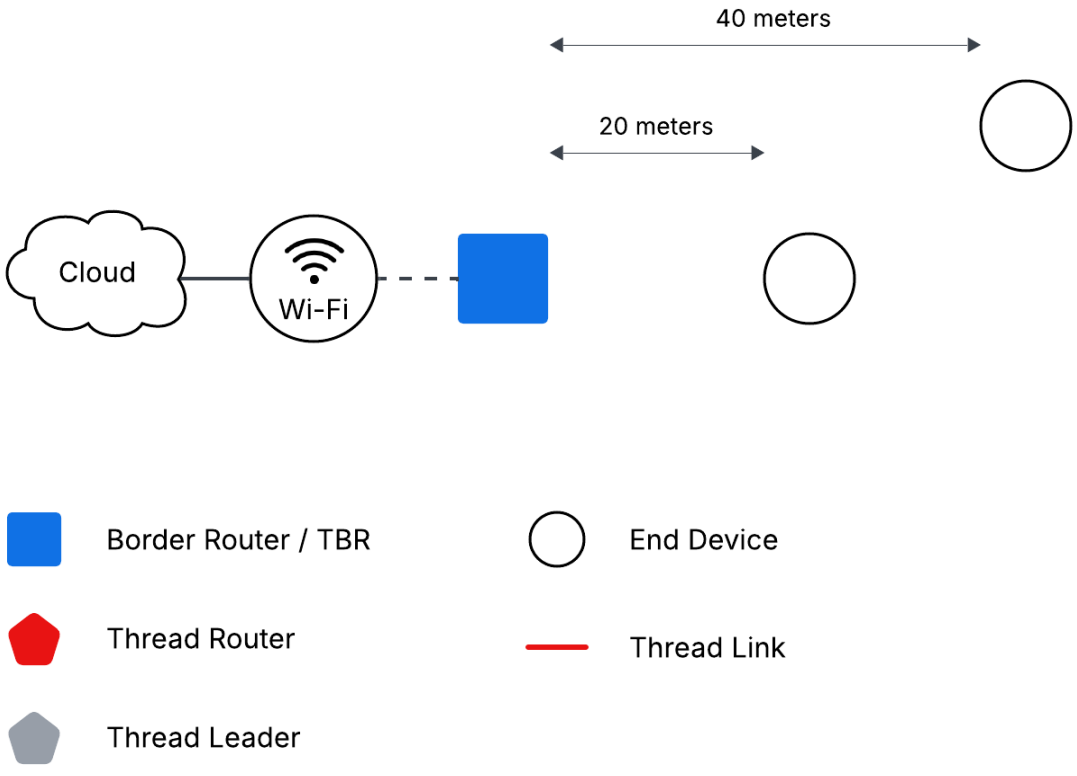

FIGURE 20. Illustration of the testing setup

Objective: Verify that devices can form a stable mesh network automatically when powered on and forward REST API calls of sensor platforms that are out of the TBR's reach.

Procedure: Place the sensor platforms far from the TBR before powering on all the devices. Then observe formation process through ESP-IDF's terminal window and using the web UI that can be enabled through TBR's settings. Observe device discovery, network joining, role assignment like router/child.

Observations: The range of which the sensor platforms could communicate with the TBR was underestimated in the initial testing setup and thus the distance between the sensor platforms and the TBR had to be increased during the testing, as shown in Figure 21.

With the initial testing setup both sensor platforms were able to connect to the TBR and send data to the cloud. After the first sensor platform was placed just far enough from the TBR, tests could be performed to determine if the first sensor platform could route the API calls of the seconds sensor platform that was out of the TBR's reach. As a result, the second sensor platform was observed to be able to connect to the TBR using the first sensor platform as an intermediary.

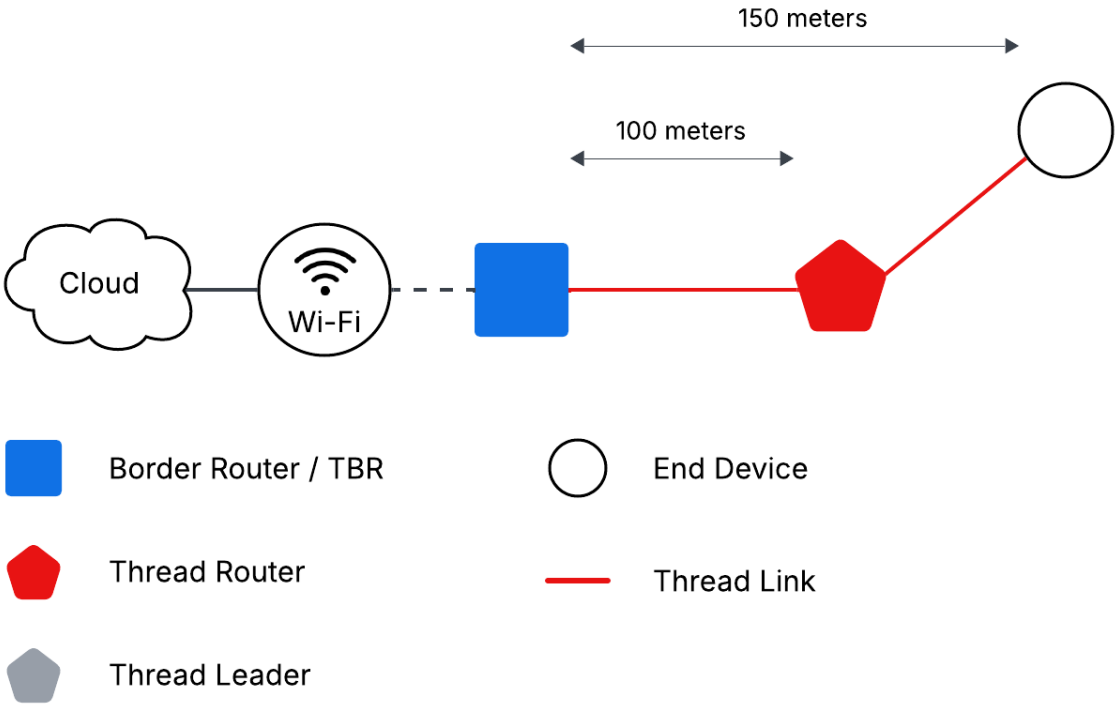

FIGURE 21. Mesh network configuration after the network creation

7.3.2 Mesh network self-healing

This testing phase is done to observe and study how the roles of the nodes are reassigned when one node is taken off from the Thread network.

Objective: Test the network's ability to recover from node failures and reassign sensor nodes roles.

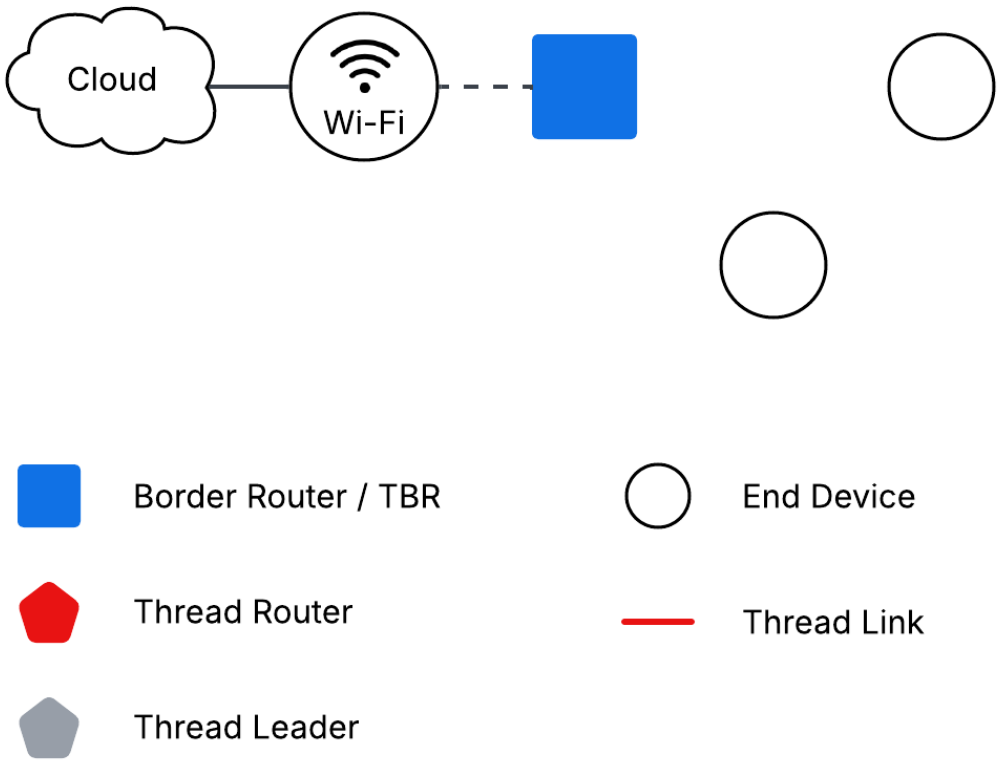

FIGURE 22. Illustration of the initial testing setup

Procedure: Two sensor nodes are placed at the reach of the TBR as shown in Figure 22. One of the sensor nodes is forced to disconnect by powering it off. Observations are made of how the network reroutes traffic and changes the other sensor node's role.

Observations: After all the devices were powered on, mesh network was automatically formed, and end devices got their roles assigned, as shown in Figure 23. Initial hypothesis was that only one of the end devices would get assigned as Thread router, but it turned out that when both are at the reach of the TBR, they both get promoted from child (end device) to router. This meant

that both sensor platforms were communicating directly with the TBR and thus even if one of them were turned off, the others communication was not affected.

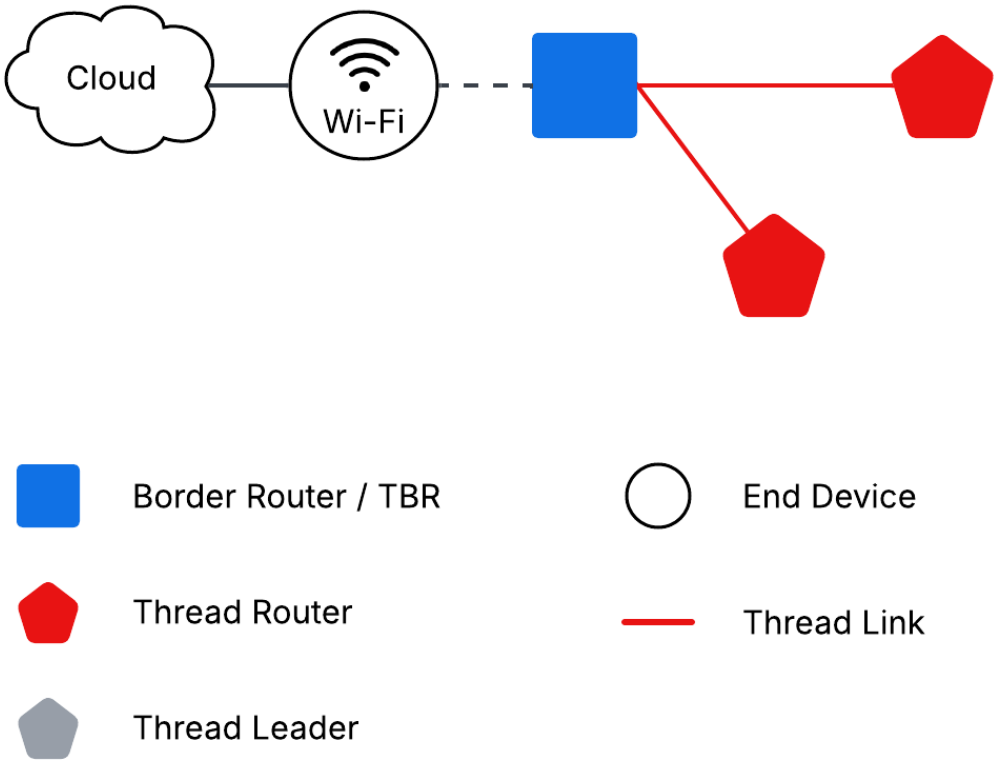

FIGURE 23. Mesh network configuration after the network creation

8 REAL-WORLD USE CASE: HOUSEPLANT MONITORING

8.1 Use case description

The idea of the house plant monitoring system came from a real-life use case shown in Figure 24, where a beginning gardener needed help monitoring the growing conditions of several different kinds of seedlings.

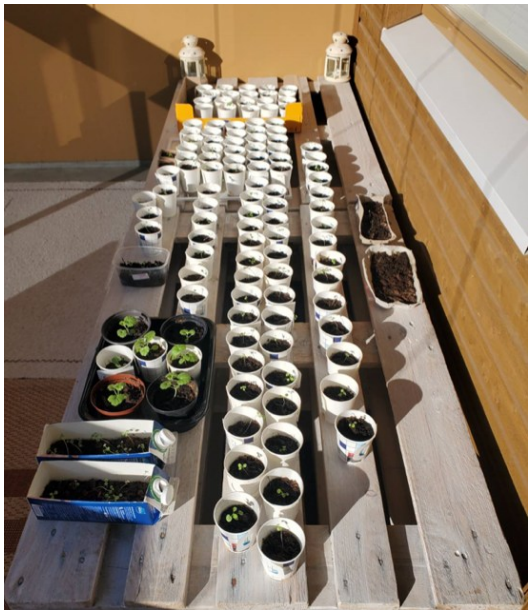

FIGURE 24. Real-world use case

With some experience, the right watering intervals and soil moisture levels would eventually be learned, but during the learning process, there is a real risk of causing the plants stress or even irreversible damage. At the beginning of the planting season, the plants are inside of the apartment at nighttime and are brought on a glazed balcony at daytime to get sunlight. In early spring the temperature needs constant monitoring. Instead of going with the trial-and-error approach, an idea of a more straightforward solution arose, and the sensor platform planning began. The aim was to build a system, where the conditions of the plants could be monitored with a web application by importing the data of the sensor platform to the internet using the Thread protocol.

8.2 Setup and deployment

The built sensor platform has two ESP32-H2 boards and a Thread Board Router. They were set up to form a Thread network, as was described in chapter 4. To build a plant monitoring system, 2 types of peripheral devices were added to the system. The Figure 25 presents the sensor system, where the two ESP32-H2 devices are set up as Routers, and the other one was set up to take the role of a Thread Leader.

FIGURE 25. Houseplant monitoring system

Two types of peripheral devices were used to monitor optimal growing conditions of the plants. One of the sensors measured the temperature and the humidity of the environment and the other one was used to measure the moisture of the soil.

Temperature & humidity sensor

The temperature and humidity sensor communicates using the I2C protocol and it measures the temperature and humidity of the surrounding environment. The temperature measurement range is from –40℃ to +125℃ with an accuracy of ±0.5℃ and the Relative Humidity measurement range is from 0% RH to 100% RH with an accuracy of ±3.0 RH%. Relative Humidity is the amount of moisture that is present in the air compared to the maximum amount the air can hold at a given temperature. These measurement ranges and accuracies are sufficient for the planned test environment. The sensors housing is made of 304 stainless steel, ensuring excellent resistance to the planned operating conditions. The sensor has a low power consumption paired with high accuracy, which makes it suitable for the plant monitoring system. (DFRobot 2025.) The sensor is connected to the ESP32-H2 board with jumper wires, as is shown in Figure 26.

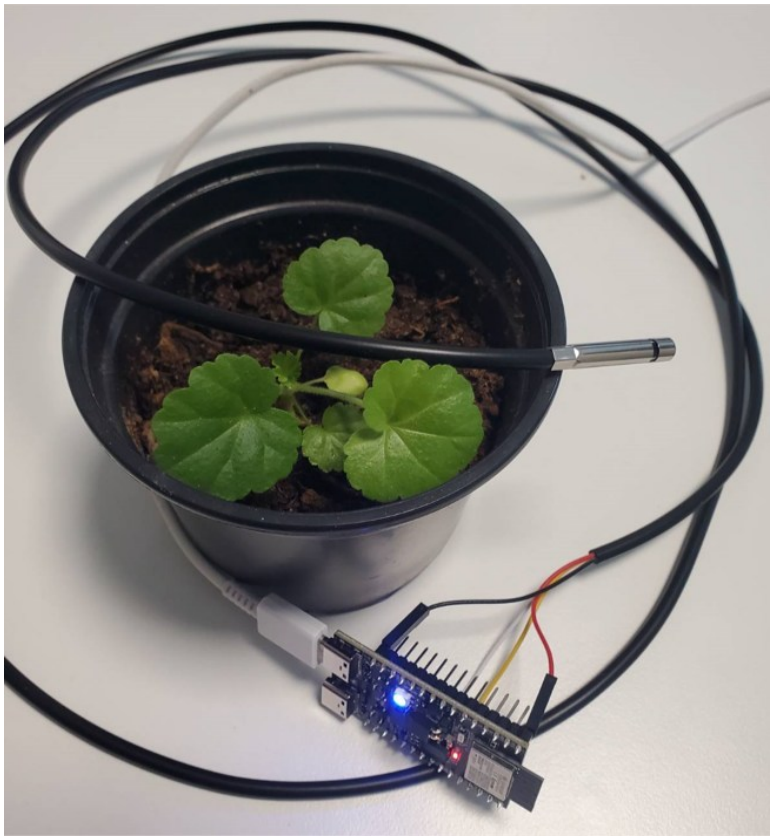

FIGURE 26. Temperature & Humidity sensor

Soil moisture sensor SEN-13322

The soil moisture sensor is for measuring the moisture in soil and similar materials. It is connected to the development board with a ground pin and VCC and it gives an analog output signal. It operates at a voltage from 3.3V to 5V. The common issue with this type of sensor is that they are exposed to a moist environment which leads to quicker wear. For this reason, the sensor has been coated in gold finish. In our project the sensor is used to measure the moisture of the soil where the plants grow, and it will be exposed to wearing conditions. This is why a good quality sensor was chosen. The sensor is a simple part of the project, yet it provides essential information for the plant monitoring system. The system will have these sensors to get measurement data from different areas of the soil. (SparkFun Electronics.) As Figure 27 shows, the sensor is connected to an ESP32-H2 board that is mounted on a bread board.

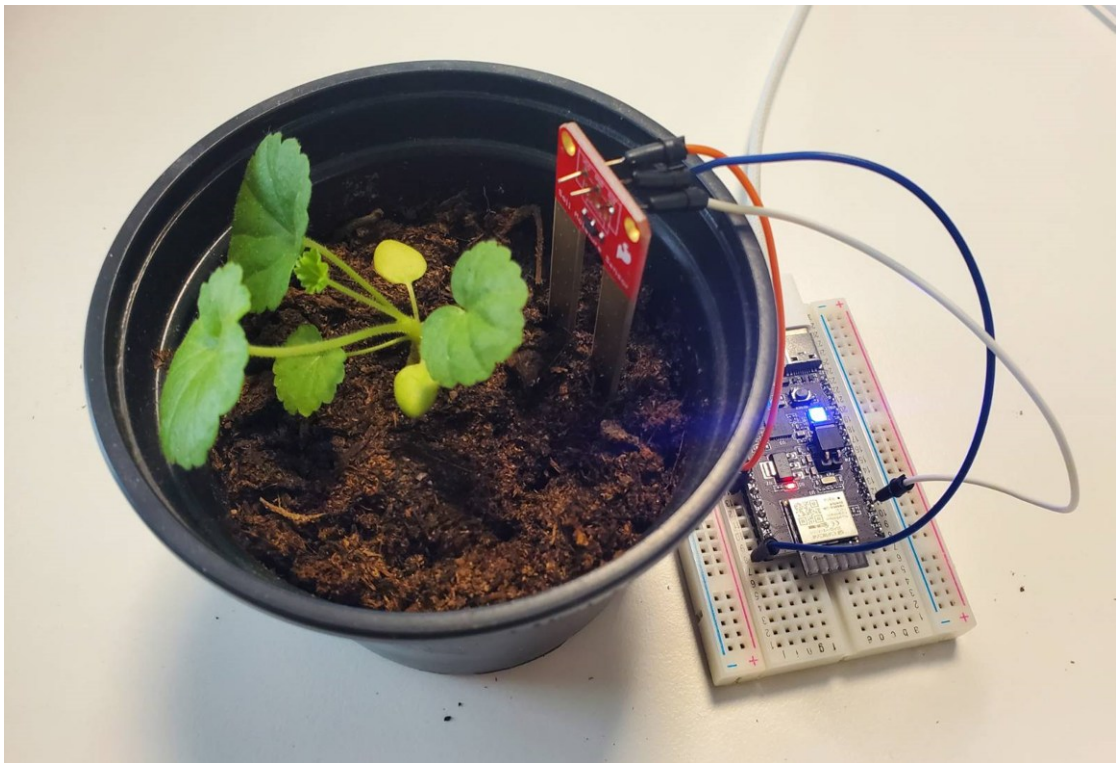

FIGURE 27. Soil moisture sensor

8.3 Summary

The basic requirements for the Thread network that are listed in chapter 3 were met and a stable Thread network was formed. The sensor platform was able to communicate wirelessly using the Thread protocol.

The Temperature and Humidity sensor data and the Soil moisture sensor data was sent to the web application. In the testing setup, the frequency of sending the data was quite high to get enough of results, but in a real user case, the interval could be adjusted. In a plant monitoring system, for example a once in an hour update interval would be sufficient.

To set the device permanently on the balcony, for example a battery could be used to power the devices. When the Thread network was tested, the signal was strong even through thick walls, so the Thread Board Router could be indoors while the sensor platform could be on the balcony. The chosen sensors have a good weather resistance, and the ESP32-H2 microcontroller supports an operational temperature range of –40°C to +85°C. (Espressif Systems 2025c.) To protect it against moisture, the ESP32-H2 board could be housed in a protective enclosure. The other option could be designing a custom PCB (Printed Circuit Board) suitable for the outdoor environment.

With an extended project timeline, more monitoring could have been done. The collected data could have been used for getting a better understanding of the correct watering cycle for different soil types. Also, the plant monitoring system could be improved with adding more sensors, for example light sensors. The web application could be set up for giving alerts, when the sensor values go out of the boundary values. In longer term, plant growth and health would be an indicator of how well the system is working.

Overall, building and testing the sensor platform was an interesting project, where new skills were learned and a lot of new ideas for future projects emerged.

9 DISCUSSION

In this project, we successfully built a functional system that utilizes the Thread protocol to connect sensor platforms to the internet. We developed both the sensor platform and the web application and were able to link them through a Thread Border Router acting as a gateway between the local mesh network and the cloud service. The basic functional requirements of the project were met, demonstrating that the core concept works. However, many features and further possibilities were left for future exploration, either by us or by other students continuing this work.

Challenges and limitations

Due to the limited time available for this project, several aspects had to be excluded from our scope. These included the manufacturing and testing of a custom PCB for the sensor platform, implementing Over-the-Air (OTA) updates, running the sensor platform on battery power, and measuring the device's energy consumption, particularly while using the ESP32 processor's deep-sleep mode.

Additionally, because of these time constraints, the testing was done manually and not automated.

Initial goal for the sensor platform was to support two-way communication, enabling real-time data transfer to both directions. However, our design choice to use REST API through HTTP requests to send data and receive commands, led us to a result where the sensor platform can send data to cloud in real-time but updates for example to the data update interval can't be pushed to the sensor platform in real-time. Instead, the sensor platform will fetch new configuration settings from the cloud whenever it is sending new data. This causes delay to possible updates and is equal to the previously set data update interval.

Lessons learned

Through this project we gained good understanding of the existing smart home systems and especially of the Thread protocol. We also got valuable experience about time management on projects like this. Viewing this retrospectively there would have been enough of a project topic in just designing and implementing the sensor platform itself and finalizing that first.

Future development and improvements

Even with a simple setup, the house plant monitoring system was an interesting real-life experiment of the Thread protocol. Due to the previously mentioned reasons, many of the planned features, as well as some that were not formally documented, had to be excluded from the scope of this project. However, should there be interest from future students, or us, to continue developing this concept, the built system offers a good starting point for future projects.

The existing sensor platform design could be used to build a bigger plant monitoring system with more devices, like light sensors and a plant watering system. An interesting possibility would also be utilizing the sensor platform for greenhouse environment monitoring or for larger agricultural use. And if the gardening aspect does not spark interest, the Thread platform introduced in chapter 4 offers flexibility for various other use cases by introducing a base for installing different kinds of peripheral devices to it. In future projects following topics could be studied:

- **The existing Thread Device types and forwarding roles**, for example Sleepy End Devices: Study how to assign the sensor platform a role in Thread network that would allow it to power off and only wake up when reading new sensor data and sending it to cloud.
- **Bigger mesh network:** Add multiple ESP32-H2 boards or other Thread devices to the Thread Network and observe what kind of a network they form and what Thread device roles they take.

- **Powered from battery:** Test how sensor platform would work when power is limited, and power optimization needs to be thought about.
- **Custom PCB:** Design a custom PCB for the sensor platform to make it more compact and provide connections e.g. for a coin battery.
- **Sensor ID:** should be generated and stored automatically by the sensor platform itself on first reboot using of UUID v4 for unique ID
- **Over-the-Air updates:** Enable users to perform OTA updates to sensor platforms for easier updates.

Topics to be explored in the future for the web application:

- **Data display:** add option to show data in the graphs for 1-, 7-, and 30-day periods at a time.
- **Alerts:** provide user tools to set thresholds for sensor values and send alerts when those are exceeded.
- **Renaming of sensor ID's:** enable user to rename their sensors for more convenient use of the web application.

The source code for this project is available on GitHub and can be accessed via the links provided in Appendix 1.

Final thoughts

Overall, this project offered interesting opportunity to study and explore modern wireless technologies and apply them in a real-life use case scenario. New skills were learned, which will surely be useful in the future.

REFERENCES

Afanef, M. 2022. Bluetooth Low Energy (BLE): A Complete Guide. URL: https://novelbits.io/bluetooth-low-energy-ble-complete-guide. Accessed: 22.4.2025.

Charlton, A. 2021. What is Zigbee, and why it's a must-have for your smart home. URL: https://www.techradar.com/news/what-is-zigbee-and-why-its-a-must-have-for-your-smart-home. Accessed: 22.4.2025

Czerniuk, M. 2025. Svelte or React? 10 Key Factors to Help You Device in 2025. SVAR Blog. URL: https://svar.dev/blog/svelte-vs-react/. Accessed: 29.3.2025.

DFRobot. 2025. I2C Temperature & Humidity Sensor (Stainless Steel Shell). URL: https://www.dfrobot.com/product-2600.html. Accessed: 21.3.2025.

Dlachenko, R. 2023. Top 7 Smart Home Protocols Compared. URL: https://lembergsolutions.com/blog/smart-home-protocols-explained. Accessed: 20.4.2025.

Espressif Systems. 2025a. ESP32-H2-DevKitM-1 User Guide. URL: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/5483/ESP32-H2-DEVKITM-1-N4.pdf. Accessed: 16.5.2025.

Espressif Systems. 2025b. ESP32-H2MINI-1 Datasheet version 1.5. URL: https://www.espressif.com/sites/default/files/documentation/esp32-mini-1_datasheet_en.pdf. Accessed: 16.5.2025.

Espressif Systems. 2025c. Get Started. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html#introduction. Accessed: 16.5.2025.

Espressif Systems. 2025d. Build and Run. URL: https://docs.espressif.com/projects/esp-thread-br/en/latest/dev-guide/build_and_run.html. Accessed: 17.5.2025.

Espressif Systems. 2025e. Hardware Platforms. URL: https://docs.espressif.com/projects/esp-thread-br/en/latest/hardware_platforms.html#wi-fi-based-thread-border-router. Accessed: 29.3.2025.

Espressif Systems. 2025f. ESP-IDF. https://github.com/espressif/esp-idf?tab=readme-ov-file#readme. Accessed: 17.5.2025.

GeeksforGeeks. 2024. What is 6LoWPAN? URL: https://www.geeksforgeeks.org/what-is-6lowpan/. Accessed: 14.4.2025.

GeeksforGeeks. 2025. What is IP Routing? URL: https://www.geeksforgeeks.org/what-is-ip-routing/. Accessed: 20.4.2025.

GeeksforGeeks. 2025. User Datagram Protocol (UDP). URL: https://www.geeksforgeeks.org/user-datagram-protocol-udp/. Accessed: 15.4.2025.

HowToGeek. 2023. What Is Wi-Fi, and How Does It Work? URL: https://www.howtogeek.com/865706/what-is-wi-fi/. Accessed: 23.4.2025.

Kumar, N. 2024. How Many IoT Devices Are There (2025-2030 Data). URL: https://www.demandsage.com/number-of-iot-devices/. Accessed: 26.3.2025.

Nordic Semiconductor. 2020. Introduction to Thread. URL: https://www.youtube.com/watch?v=EDseO8GY9Cg. Accessed: 25.3.2025.

OpenThread. 2025. Node Roles and Types. URL: https://openthread.io/guides/thread-primer/node-roles-and-types. Accessed: 19.4.2025.

OpenThread. 2025. Border Router. URL: https://openthread.io/guides/border-router. Accessed: 1.4.2025.

Poole I. IEEE 802.15.4. Standard: a tutorial / primer. URL: https://www.electronics-notes.com/articles/connectivity/ieee-802-15-4-wireless/basics-tutorial-primer.php. Accessed: 14.4.2025.

RFWirelessWorld 2025. Wireless Communication Protocols: Z-Wave vs. Thread vs. Bluetooth vs. WiFi vs. ULE vs. EnOcean vs. Zigbee. URL: https://www.rfwireless-world.com/terminology/iot/wireless-communication-protocols-comparison. Accessed: 20.4.2025.

SparkFun Electronics. SparkFun Soil Moisture Sensor. URL: https://www.sparkfun.com/sparkfun-soil-moisture-sensor.html. Accessed: 21.3.2025.

Thread Group 2020. Thread Network Fundamentals. URL: https://www.threadgroup.org/Portals/0/documents/support/Thread%20Network%20Fundamentals_v3.pdf. Accessed: 14.4.2025.

Thread Group. 2025. What is Thread? URL: https://www.threadgroup.org/What-is-Thread/Overview. Accessed: 14.4.2025.

Walsh, D. 2025. Zigbee vs Thread: Which is Better? URL: https://www.smarthomeperfected.com/zigbee-vs-thread/. Accessed: 22.4.2025.

Z-wave Alliance. 2025. Technology Overview. URL: https://z-wavealliance.org/technology-overview/. Accessed: 20.4.2025.

APPENDICES

Appendix 1 Source code repositories

SOURCE CODE REPOSITORIES APPENDIX 1

The source code for this project is available in the GitHub repositories linked below. Instructions for setting up the system and continuing development are provided in the repositories README files.

https://github.com/tm-thesis-2025/sensor-platform

https://github.com/tm-thesis-2025/web-application