VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Internet: office@ecs.vuw.ac.nz

SWEN 428 IoT Project Report

Oscar W. Ashburn 300679727

Supervisor: NOT STATED

Submitted in partial fulfilment of the requirements for
Master of Computer Science.

Abstract

This document presents the design, implementation and evaluation of a BLE-based IoT system for environmental monitoring, comprising of an always-on gateway and sensor nodes. The gateway continuously scans for sensor nodes within range, establishes BLE connections, and queries them for environmental data, specifically temperature and humidity.

Revision History

Name	**Changes**	**Version**	Date
Oscar Ashburn	Initial copy	1.0.0	09/09/2025
Oscar Ashburn	Added in depth look at each part of the system	2.0.0	09/09/2025
Oscar Ashburn	Tidied up and changed focus to explain GATT and GAP setup	3.0.0	11/09/2025
Oscar Ashburn	Added software design for working with the hardware.	3.1.0	15/09/2025
Oscar Ashburn	Revised hardware to reflect the usage of RIOT library.	3.2.0	15/09/2025
Oscar Ashburn	Revised BLE handler callbacks.	3.3.0	20/09/2025
Oscar Ashburn	Added source code structure to report body	3.4.0	23/09/2025
Oscar Ashburn	Added program flow diagram.	3.5.0	23/09/2025
Oscar Ashburn	Added discussion of power and latency optimisation	3.6.0	23/09/2025
Oscar Ashburn	Explained design decision	3.6.0	26/09/2025
Oscar Ashburn	Results and testing	4.0.0	26/09/2025
Oscar Ashburn	Discussion and Conclusion	4.1.0	01/10/2025
Oscar Ashburn	Source code added to appendix	4.2.0	03/10/2025

Contents

Chapter 1

Introduction

The goal of this project was to develop a BLE-enabled gateway capable of discovering and reading environmental sensor data. The sensor nodes need to provide temperature (°C) and humidity (%) readings from a Hts221 sensor.The key challenge is to get the sensor discovery below 100ms.

This report is structured to provide an overview of the project. The design section details the architecture and communication protocols of both the gateway and sensor nodes. The evaluation section explains the methodology used to measure system performance, including discovery latency and data retrieval success rates. The results section presents the findings from these evaluations, highlighting system strengths and limitations. Finally, the conclusion discusses the overall effectiveness of the system and suggests potential improvements for future work, including optimizations for real-time monitoring and power efficiency.

1.0.1 Scope

The core application is executed on the gateway, where it manages interactions with the sensor nodes. The BLE protocol is utilized to discover sensors, establish connections, and request environmental data, specifically temperature and humidity. The format of the communication between the gateway and the sensors may be defined as part of the design. However, each sensor response is required to include the following information: a Unix timestamp, the sensor reading value, and the corresponding unit of measurement. For temperature the unit must be Celsius, and for humidity the unit must be Percent. Error messages may also be defined and formatted by the gateway to be returned to the application when necessary.

The gateway node is required to remain powered on at all times. It must be capable of scanning for nearby sensor nodes, establishing connections with them, and retrieving data through the BLE interface.

The sensor nodes are also required to remain powered on at all times. Each sensor must advertise its presence and the type of data it provides, such as temperature or humidity, through the BLE interface. This enables discovery by the gateway node. Once discovered, the sensor must allow the gateway to establish a connection so that data can be read.

A requirement for discovery latency optimisation is imposed. The system must be configured so that discovery latency is less than 100 milliseconds in at least 50 percent of the cases. To meet this requirement, BLE link-layer parameters such as the scan window, scan interval, and advertising interval may be adjusted. At least 100 measurements must be collected to evaluate performance, and at least 50 of these measurements must record a latency of less than 100 milliseconds.

Chapter 2

Design

2.1 Hardware

For this project, I used the Nordic nRF52-DK development board [1], which features BLE capabilities. The board is equipped with an HTS221 temperature and humidity sensor, enabling environmental data collection. It was hosted on the FIT IoT-LAB testbed[2], providing a remote, controlled environment for experimenting with sensor discovery, data collection, and BLE communication.

2.2 nRF52DK

The nRF52 Development Kit (nRF52DK) is a single-board development platform based on the Nordic Semiconductor. The DK is designed for prototyping low-power wireless applications[3].

2.2.1 Key Features

- Bluetooth Low Energy (BLE).

2.2.2 Role in the Project

- Serves as the hardware platform for both sensor nodes and the gateway
- Provides BLE radio for advertising and scanning

2.3 HTS221 Temperature & Humidity Sensor

The Hts221 sensor attached to the nrf52DK can be queried via the RIOT OS driver[4].

2.3.1 Role in the Project

- Read Temperature and Humidity

2.4 System Overview

A distributed sensor network based on Bluetooth Low Energy (BLE) for collecting environmental data (temperature, humidity) from multiple low-power sensor nodes using a **single-hop topology**. A central gateway manages sensor discovery, connection, data collection, data processing and user feedback.

2.4.1 Core Gateway Application

- Scans for BLE sensors of a specific type (temperature or humidity).
- Connects to a discovered sensor and reads characteristic values via GATT.
- Stores readings in `sensor_response_t` including:
  - Unix timestamp
  - Sensor reading value
- Handles discovery latency measurement and errors (timeouts, failed connections).
- Provides shell commands (`get_temp`, `get_humid`, `help`).

2.4.2 Gateway Node Responsibilities

Always on and scanning for BLE sensors. Scans specifically for a Temperature UUID or Humidity UUID defined by Bluetooth SIG[5].

- Handles BLE GAP events (connect, disconnect).

Performs service and characteristic discovery, reading the Environmental Sensing Service (ESS)[6] if available. Measures and logs discovery latency.

2.4.3 Sensor Node Responsibilities

- Always on and advertising BLE presence.
- Advertises sensor type (temperature or humidity) via 16-bit UUIDs.
- Responds to GATT read requests with value and timestamp.

2.5 Communication Protocol

The sensor and packets a timestamp and 16 bit integer and sends the packet data across Bluetooth. The datatype to store the reading just needs to be big enough for 16 bits as processing and display happens on the gateway application.

```
typedef struct packet_t {
    int16_t reading;
    uint32_t timestamp;
} packet_t;
```

2.6 Sequence of Operations

The sequence of operations are as follows. A shell command triggers a BLE query, generating a request ID with a timestamp and starting the scanner with a configured interval/window; upon receiving an advertisement, the system checks the UUID, records discovery latency, and initiates a connection, after which it discovers services and characteristics, reads the GATT characteristic, parses the reading with a timestamp, updates the sensor_response_t structure, prints the result, disconnects, and relies on a timeout thread to ensure the scanner does not run indefinitely.

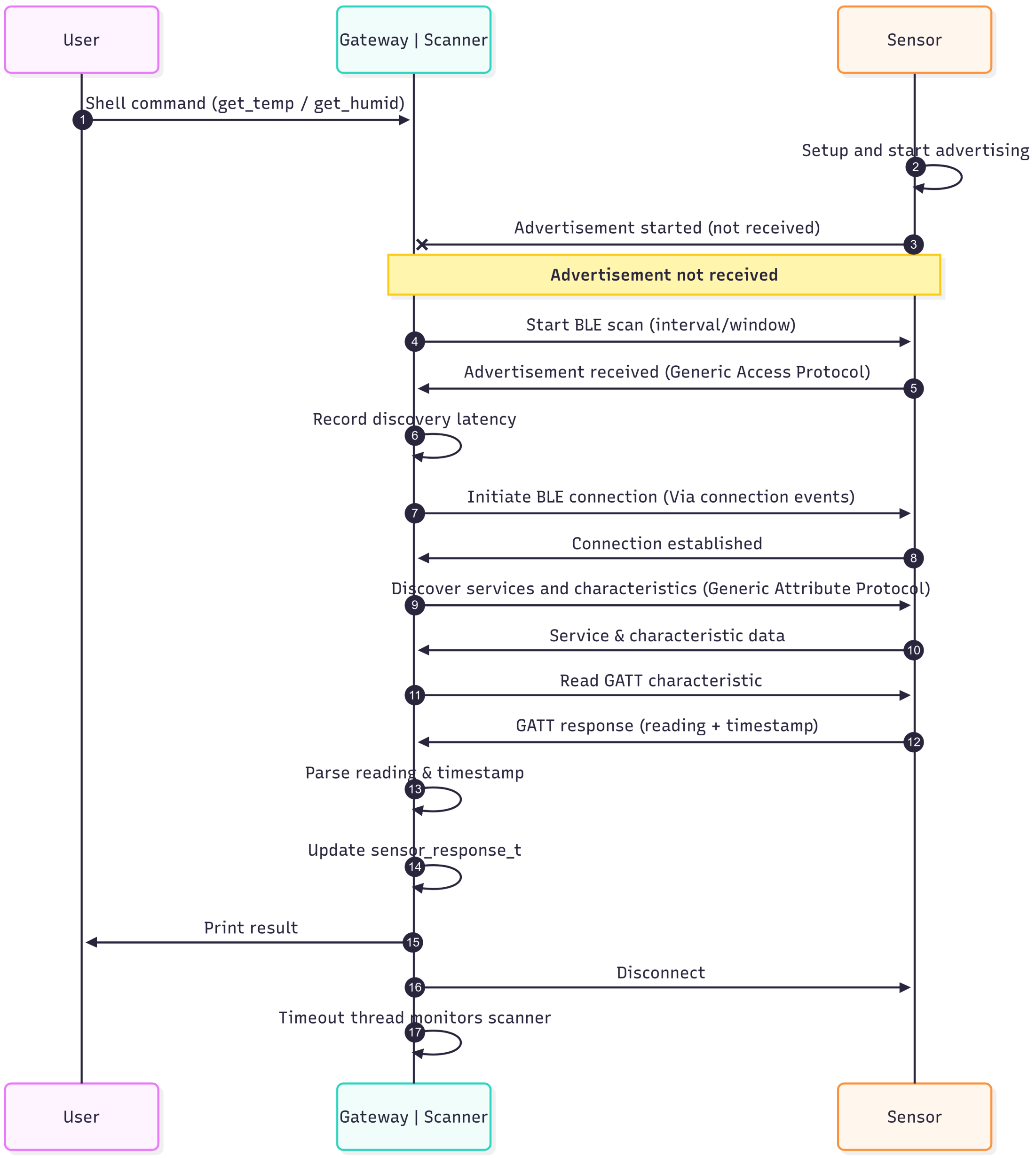
User
Gateway | Scanner
Sensor
Shell command (get_temp / get_humid)
Setup and start advertising
Advertisement started (not received)
Advertisement not received
Start BLE scan (interval/window)
Advertisement received (Generic Access Protocol)
Record discovery latency
Initiate BLE connection (Via connection events)
Connection established
Discover services and characteristics (Generic Attribute Protocol)
Service & characteristic data
Read GATT characteristic
GATT response (reading + timestamp)
Parse reading & timestamp
Update sensor_response_t
Print result
Disconnect
Timeout thread monitors scanner
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

Figure 2.1: Execution Flow

Chapter 3

Gateway Node

The Gateway Node uses RIOT os Nimble ports and to communicate with the Sensor Node.
The gateway uses various information about a current request such as the following.

- **request_id [char[32]]**: Unique identifier for the request.
- **success [bool]**: Indicates whether the sensor query succeeded.
- **value [double]**: The numeric value of the sensor reading.
- **timestamp [uint32_t]**: Unix timestamp representing when the reading was taken.
- **unit [char[16]]**: Unit of the reading, e.g., "Celsius" for temperature or "Percent" for humidity.
- **discovery_latency_ms [uint32_t]**: Time in milliseconds taken to discover the sensor.
- **error_message [char[64]]**: Description of any error that occurred if the query failed.

3.0.1 Scanning

The Gateway Node searches for the uuids **TEMPERATURE_CHARACTERISTIC_UUID 0x2A6E** and **HUMIDITY_CHARACTERISTIC_UUID 0x2A6F** to determine it has discoverers the viable sensor.

These UUIDs are defined by the Bluetooth SIG as part of the Environmental Sensing Service (ESS) standard.

3.0.2 Data Reading

Once the Gateway Node has discovered a sensor by matching the target UUID, it establishes a BLE connection and performs service and characteristic discovery. The Gateway reads the sensor's characteristic values using the GATT protocol. Each reading includes:

- **Unix timestamp:** The time at which the reading was taken.
- **Sensor value:** Either temperature in Celsius or humidity in Percent.

If the sensor fails to respond, the Gateway handles the error by recording an error message and stopping the scan. Each successful reading also records the discovery latency, which is the time elapsed from scan start to sensor detection.

3.0.3 Source File Overview

The Gateway Node BLE handler and main application are implemented across `gateway.c` and `ble_handler.c`. Below is a summary of the main functions and their responsibilities.

- **generate_request_id(char *request_id, size_t size)**
  Generates a unique request ID for each sensor query by combining a counter with the current timestamp (milliseconds). Ensures each sensor reading can be traced uniquely.

- **get_scan_elapsed_ms(void)**
  Returns the elapsed time (in milliseconds) since scanning started. Used for computing sensor discovery latency.

- **ble_query_sensor(const int sensor_type)**
  Initiates a BLE scan for the specified sensor type (`SENSOR_TEMP` or `SENSOR_HUM`). Prepares a new `active_response_t` structure, sets the unit and target type, and starts the NimBLE scanner. Handles scan timeout and error conditions.

- **check_scan_timeout(void)**
  Periodically called to determine if the scan has exceeded the predefined timeout. If so, stops the scanner, updates the response with failure, and logs an error message.

- **scan_cb(...)**
  Scanner callback invoked for each BLE advertisement. Checks if the advertised UUID matches the desired sensor, stops scanning, and initiates a connection to the sensor. Computes discovery latency and updates `active_response`.

- **gap_event_cb(struct ble_gap_event *event, void *arg)**
  Handles BLE GAP events such as connection and disconnection. On connection, it triggers service discovery; on disconnection, it resets connection state.

- **svc_disc_cb(...)**
  GATT service discovery callback. Searches for the Environmental Sensing Service (ESS) [6] on the connected device. Initiates characteristic discovery if the service is found.

- **chr_disc_cb(...)**
  Characteristic discovery callback. Determines which characteristic to read based on the sensor type. Initiates a GATT read for either temperature or humidity characteristic.

- **gatt_read_cb(...)**
  Called when a GATT read operation completes. Extracts the sensor reading and timestamp, converts raw data to appropriate units, updates the `active_response_t` structure, prints the reading, and disconnects from the sensor.

- **cmd_get_temp(int argc, char **argv)** and **cmd_get_humid(int argc, char **argv)**
  Shell command handlers that trigger a temperature or humidity query respectively by calling `ble_query_sensor()`.

- **timeout_thread(void *arg)**
  Runs in a dedicated thread to continuously monitor for scan timeouts by calling **check_scan_timeout()** every x ms where x is the desired timeout.

- **main(void)**
  Entry point of the Gateway application. Initializes the BLE scanner, starts the timeout thread, and runs the shell interface to accept user commands. Sets up scanner parameters and binds the scanner callback.

3.0.4 Design Notes

- The Gateway Node is designed to perform BLE scanning, service and characteristic discovery, and sensor reads in a non-blocking, asynchronous manner using callbacks.
- `active_response_t` holds all sensor response data, including value, unit, timestamp, and discovery latency, ensuring all relevant information is available after a query.
- Compile-time constants such as **TEMPERATURE_CHARACTERISTIC_UUID** and **HUMIDITY_CHARACTERISTIC_UUID** enable the application to select the correct characteristic without overuse of runtime branching.
- Scan timeouts and connection errors are handled gracefully, updating the sensor response and allowing the application to continue operating without blocking.

DEBUG Flag: The DEBUG flag in the Makefile was used to control debug output for the Gateway Node. When DEBUG is set to 1, the application prints detailed information about BLE scanning, connections, service and characteristic discovery, GATT read callbacks, and sensor responses. This facilitated development and troubleshooting by providing visibility into the BLE communication and sensor discovery processes. Setting DEBUG to 0 suppresses these messages, which reduces console clutter and improves runtime performance. This was also intentionally a compile time choice as realistically a customer wouldn't need access to the debug version.

3.0.5 Further Notes

The implemented solution is single threaded due to time constraints.

Chapter 4

Sensor Node

The sensor interface of the Gateway Node is implemented using two main modules: `hts221.c` for sensor initialization and readings, and `application.c` for BLE GATT and GAP handling. The design ensures that each sensor is initialized and configured only once, with key parameters determined at compile time using preprocessor macros (e.g., `SENSOR_TYPE`).

4.0.1 Source File Overview

- **hts221.c** – Responsible for HTS221 sensor initialization, power management, and reading temperature and humidity values.
- **sensor.c** – Handles the main application logic, including GATT and GAP callbacks, BLE advertising, and characteristic access.

The following key functions were implemented to allow seamless commnnication and data access.

- **create_sensor()** – Allocates memory, initializes the HTS221 sensor with parameters, and powers it on. Returns a pointer to the sensor object.
- **destroy_sensor()** – Powers off the sensor and frees allocated memory.
- **query_temperature(hts221_t *dev, int16_t *temperature)** – Performs a one-shot temperature read from the sensor. Returns 0 on success.
- **query_humidity(hts221_t *dev, uint16_t *humidity)** – Performs a one-shot humidity read from the sensor. Returns 0 on success.
- **gatt_svr_chr_access_temperature()** – BLE GATT callback for reading the temperature characteristic, returning a packet with reading and timestamp.
- **gatt_svr_chr_access_humidity()** – BLE GATT callback for reading the humidity characteristic, returning a packet with reading and timestamp.
- **gap_event_handler()** – Handles BLE GAP events such as connection, disconnection, and advertising completion.
- **main()** – Initializes the sensor, registers GATT services, configures GAP, sets advertising fields, and starts BLE advertising.

4.0.2 Design Notes

- The system is designed to initialize the sensor only once and use compile-time configuration to select the active sensor characteristic.
- Timestamping of sensor readings is performed in milliseconds using `ztimer_now(ZTIMER_MSEC)`.
- BLE services and characteristics are added during startup, and runtime branching is minimized.

```
#ifndef SENSOR_TYPE
#define SENSOR_TYPE 0
#endif

#if SENSOR_TYPE == 0
    #define SENSOR_CHAR_UUID 0x2A6E
    #define SENSOR_ACCESS_CB gatt_svr_chr_access_temperature
#elif SENSOR_TYPE == 1
    #define SENSOR_CHAR_UUID 0x2A6F
    #define SENSOR_ACCESS_CB gatt_svr_chr_access_humidity
#else
    #error "Unknown SENSOR_TYPE"
#endif
```

This approach ensures that the correct GATT characteristic and callback are selected without runtime branching, making the application efficient and safe.

4.1 Discovery Latency Optimisation

The following BLE parameters are configurable and influence device discovery latency: scan interval, scan window, and advertising interval. For evaluation, discovery latency is measured for each successful detection, targeting a median latency below 100 ms. A minimum of 100 samples is collected to ensure statistical significance.

Theoretical Power Analysis of Sensor Nodes

Even without access to the physical hardware, it is possible to provide a theoretical estimate of a BLE device's average power consumption. The average power over a connection interval can be expressed as:

$$P_{avg} = \frac{P_{active} \cdot t_{active} + P_{sleep} \cdot (T - t_{active})}{T}$$

where:

- P_{active} = power during active operations
- t_{active} = active time per interval
- P_{sleep} = power during sleep or low-power mode
- T = total interval or connection period

Discussion: Although we cannot measure these parameters directly, the formula provides insight into how the device's duty cycle and sleep strategy affect average power. This highlights key trade-offs in BLE design, such as:

- Minimizing active time to reduce power consumption.
- Balancing connection interval length to optimize battery life without excessively increasing discovery latency.

Even as a purely theoretical model, this analysis gives the report a more technical and "engineered" feel, showing an understanding of **power-performance trade-offs** in BLE systems. Though as it is the discovery latency requirements were satisfied

Chapter 5

Evaluation

5.1 Evaluation Setup

The evaluation was conducted using a custom program, `evaluation.c`, which extended the gateway application with two additional shell commands: `temp_eval` and `humid_eval`. Each of these commands repeatedly queried the sensor at a fixed rate of one request every 0.5 seconds, chosen to be slightly faster than a human user could realistically issue queries. This setup ensured sufficient measurement data while avoiding excessive system load.

Objective: To evaluate the performance of the gateway and sensor system under repeated queries at a rate slightly faster than human interaction.

Factors and Levels

Factor	Levels	Description
Command type	`temp eval`, `humid eval`	Two shell commands added to the gateway to query temperature and humidity.
Query interval	0.5 s	Fixed interval between repeated sensor queries.
Gateway scan parameters	Scan interval: 30 ms, Scan window: 30 ms	Configures how often the gateway listens for advertisements.
Sensor advertising	`nimble_autoadv()` default	Default BLE advertising configuration from the sensor.

Table 5.1: Evaluation factors and their levels.

On the sensor side, advertising was performed using the `nimble_autoadv()` default configuration.

Evaluation Scenarios

Scenario	Command	Query Interval	Scan Interval	Scan Window
1	temp eval	0.5 s	30 ms	30 ms
2	humid eval	0.5 s	30 ms	30 ms

Table 5.2: Evaluation scenarios.

Rationale:

- Test system performance under slightly faster-than-human query rates.
- Fixed scan and advertising parameters isolate the effect of command type on performance and measurement reliability.

Chapter 6

Results and Discussion

6.0.1 Humidity Sensor

- Total runs: 100
- Successful readings: 100
- Success rate (fast discovery): 91.0%
- Average discovery latency: 55.48 ms
- Minimum discovery latency: 7 ms
- Maximum discovery latency: 107 ms
- Evaluation status: Completed

6.0.2 Temperature Sensor

- Total runs: 100
- Successful readings: 100
- Success rate (fast discovery): 86.0%
- Average discovery latency: 58.51 ms
- Minimum discovery latency: 5 ms
- Maximum discovery latency: 303 ms
- Evaluation status: Completed

Comparison Table

Metric	Humidity Sensor	Temperature Sensor
Total runs	100	100
Successful readings	100	100
Average discovery latency	55.48 ms	58.51 ms
Minimum discovery latency	7 ms	5 ms
Maximum discovery latency	107 ms	303 ms
Readings under 100 ms	91 (91.0%)	86 (86.0%)

The aggressive scanning allowed for very quick discovery times which met the requirements in the specification. Though there was not sufficient time, further testing with different parameters (scan interval and window) on the gateway node and advertise interval on the sensor node would need to be tested to optimise power consumption.

Chapter 7

Conclusions and Recommendations

7.1 Conclusions

This paper discussed the design and implementation of a BLE sensor network which successfully implemented temperature and humidity sensor nodes and a gateway node that continue scans for them on request. This paper also discusses the discovery latency optimisation requirement and theoretical power consumption measurements as well as evaluation and testing.

7.2 Recommendations

This section discusses recommendations for how future work could be improved such as the design or implementation.

7.2.1 BLE Mesh

Future work could explore Bluetooth Low Energy Mesh, which enables many-to-many device communication. This would extend the system beyond point-to-point links, allowing scalability to larger networks and supporting applications such as smart environments or distributed sensing.

7.2.2 Configurable Discovery Intervals

Future work could allow configurable scan and advertising intervals, enabling developers to balance discovery latency against energy consumption. Shorter intervals would support rapid device discovery and interaction, while longer intervals suit low-power, long-term sensing. Adaptive schemes, such as increasing scan frequency after an event trigger, could further improve efficiency and applicability across different deployment scenarios. As a result this would also improve ease of testing and evaluation as recompiling (A time consuming task) would be not needed.

7.2.3 Over-the-Air Update (OTAU)

The nRF52 series supports over-the-air firmware updates, which could be integrated in future work to simplify deployment and maintenance. This would enable devices to receive software patches or feature upgrades without requiring physical access.

Bibliography

[1] F. IoT-LAB, "Nordic nrf52dk." `https://iot-lab.github.io/docs/boards/nordic-nrf52dk/`. Accessed: 2025-09-26.

[2] F. IoT-LAB, "The very large scale iot testbed." `https://www.iot-lab.info`. Accessed: 2025-09-26.

[3] N. Semiconductor, "nrf52 dk development kit." `https://www.nordicsemi.com/Products/Development-hardware/nRF52-DK`. Accessed: 2025-09-26.

[4] R. OS, "St hts221 digital humidity sensor." `https://doc.riot-os.org/group__drivers__hts221.html`. Accessed: 2025-09-26.

[5] Bluetooth SIG, "Assigned numbers: Gatt characteristics and descriptors page 32," 2025. Provides UUID values for characteristics such as Temperature, Humidity, and others used in the Environmental Sensing Service.

[6] Bluetooth SIG, "Assigned numbers: Gatt characteristics and descriptors page66," 2025. Provides UUID values for characteristics such as Temperature, Humidity, and others used in the Environmental Sensing Service.