AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

Projekt dyplomowy

Opracowanie aplikacji protokołu Thread na platformie nRF5x
Development of Thread application using nRF5x platform

Autor: *Paweł Tymoczko*
Kierunek studiów: *Elektronika i Telekomunikacja*
Opiekun pracy: *dr inż. Łukasz Krzak*

Kraków, 2024

Spis treści

Wstęp

Wraz ze wzrostem popularności oraz stopnia skomplikowanie systemów automatyki domowej i budynkowej, coraz bardziej uwypuklają się problemy związane z integracją poszczególnych technologii komunikacji bezprzewodowej, takich jak Zigbee, Bluetooth, Wi-Fi. Dodatkowo zwiększa się zapotrzebowanie na łączenie tego typu systemów z chmurą, jak również rosną wymagania dotyczące bezpieczeństwa sieci kratowych (ang. Mesh) [1]. Jedną z odpowiedzi rynku na wymienione problemy jest opracowany przez Thread Group protokół Thread.

Protokół Thread to protokół sieci kratowych, dedykowany dla urządzeń IoT (ang. *Internet of Things*) małej mocy oraz ograniczonym paśmie, który został zaprojektowany na podstawie sprawdzonych technologii, takich jak IEEE 802.15.4 oraz 6LoWPAN (ang. *IPv6 over Low-Power Wireless Personal Area Network*). Thread opiera się o IPv6 (ang. *Internet Protocol, Version 6*), co znacznie ułatwia połączenie z chmurą lub szeroko pojętym Internetem. Ponadto standard ten definiuje mechanizmy poprawiające bezpieczeństwo sieci kratowych, sprawiając, że urządzenia w łatwy i bezpieczny sposób są w stanie dołączyć do istniejącej sieci Thread.

Zastosowanie stosu Thread nie tylko niesie ze sobą bezpieczeństwo oraz mechanizmy ułatwiające prostą integrację w systemach automatyki budynkowej, ale może stanowić alternatywę dla systemów opartych na Zigbee lub Bluetooth Mesh, ze względu na oferowaną niezawodność. Jak pokazuje analiza przeprowadzona przez Silicon Labs [2] w ramach porównania 3 protokołów IoT, Thread wykazuje niższe czasy opóźnień w stosunku do reszty.

Pomimo kompatybilności protokołu Thread z urządzeniami opartymi o IEEE 802.15.4, nie jest możliwe, aby zaimplementować stos protokołów Thread we wszystkich urządzeniach LR-WPAN (z ang. *Low Range Wireless Personal Area Network*). Czynnikiem ograniczającym są zasoby sprzętowe urządzeń, takie jak pamięć, co wynika z potrzeby wsparcia warstwy sieciowej oraz funkcjonalności z nią związanych.

Próba weryfikacji, jak protokół Thread o szeregu cech, korzyści i ograniczeń wymienionych powyżej znajduje zastosowanie w systemie automatyki domowej i budynkowej, stanowi motywację do powstania tej pracy.

Cel Pracy

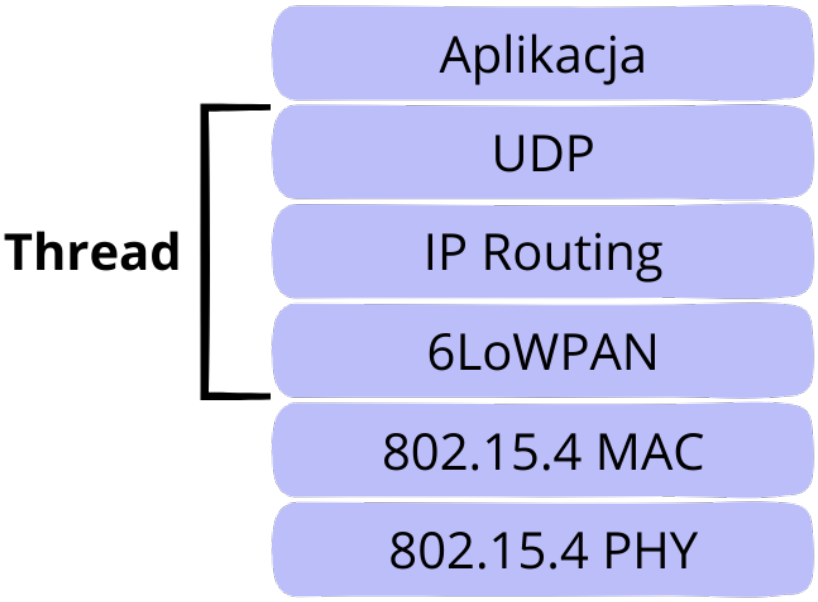
Celem pracy jest opracowanie prototypu przykładowego systemu automatyki domowej lub budynkowej, wykorzystującego stos protokołów Thread. System został zaimplementowany na jednej z platform Nordic Semiconductor serii nRF5x, z wykorzystaniem zestawu narzędzi dostarczonych przez producenta. Etapem końcowym projektu stanowi demonstracja działania systemu opartego na co najmniej 5 urządzeniach serii nRF5x oraz analiza zasobów sprzętowych pamięci, niezbędnych do uruchomienia stosu Thread na wykorzystanej platformie.

1. Wprowadzenie do Thread

Niniejszy rozdział ma na celu przybliżenie informacji dotyczącej Thread, poprzez przedstawienie i omówienie stosu protokołów, wyszczególnienie urządzeń i ról, w jakich pracują węzły sieci Thread oraz zaprezentowanie wybranych, podstawowych mechanizmów, przydatnych do zrozumienia zagadnienia sieci Thread. Rozważania w poniższym rozdziale oparte zostały o publiczną dokumentację Thread [3].

1.1. Stos protokołów

Stos protokołów Thread obejmuje warstwę sieciową oraz transportową Modelu OSI (ang. *ISO Open Systems Interconnection Reference Model*) i jest dedykowany dla urządzeń opartych o warstwę fizyczną oraz warstwę łącza danych standardu IEEE 802.15.4. Na Rysunku 1.1. przedstawiono warstwy Thread oraz wdrożone w nich protokoły komunikacyjne.



Rys. 1.1. Stos protokołów Thread.

1.1.1. Warstwa Fizyczna oraz Warstwa Łącza Danych

Protokół Thread jest stworzony dla urządzeń, których 2 pierwsze warstwy modelu OSI są oparte o standard definiujący LR-WPAN w wersjach IEEE 802.15.4-2006 lub IEEE 802.15.4-2015. Jednakże nie jest wymagane, aby urządzenia Thread funkcjonowały w sieciach LR-WPAN. Ponadto protokół Thread nie wykorzystuje zdefiniowanych w IEEE 802.15.4 ról, takich jak:

- FFD (ang. *Full-Function Device*),
- RFD (ang. *Reduced-Function Device*),
- Koordynatora PAN (ang. *Personal Area Network Coordinator*).

1.1.2. Warstwa Sieciowa

W warstwie 3. Modelu OSI Thread implementuje protokół IPv6.

W celu umożliwienia przesyłania pakietów IPv6 nad LR-WPAN wprowadzono warstwę adaptacyjną 6LoWPAN. Usprawnienie to pozwala na enkapsulacje pakietów IPv6 do IEEE 802.15.4 MSDU (ang. *MAC Service Data Unit*) oraz definiuje mechanizmy kompresji nagłówków IPv6, fragmentacji i składania pakietów IPv6, odpowiednio do oraz z IEEE 802.15.4 MSDU.

Do trasowania (ang. *routing*) Thread wykorzystuje zdefiniowany w rozdziale 5.9 dokumentacji [3] protokół rutingu typu distance vector.

1.1.3. Warstwa Transportowa

Do komunikacji między węzłami w warstwie transportowej sieć Thread używa UDP (ang. *User Datagram Protocol*). Protokół ten jest wymagany przy implementacji stosu Thread, ponieważ jest wykorzystywany w procesach sygnalizacyjnych oraz mechanizmach zarządzania siecią Thread.

Wdrożenie TCP (ang. *Transport Control Protocol*) nie jest obligatoryjne. Natomiast w przypadku potrzeby uwzględnienia TCP, Thread definiuje wskazówki do efektywnej implementacji tego protokołu w rozdziale 6.2 dokumentacji [3].

1.2. Typy urządzeń oraz role

Niniejsza sekcja ma na celu objaśnienie podstawowych ról węzłów w sieci Thread oraz terminologii, wykorzystywanej w opisywaniu mechanizmów sieci.

1.2.1. Podstawowe role urządzeń

W celu pełnej klasyfikacji węzłów w sieci Thread dokonano ich podziału ze względu na umiejętność przekazywania (ang. *forwarding*) pakietów oraz opisano ich charakterystykę:

1. Ruter (ang. *router*):

- posiada możliwość przekazywania pakietów między węzłami sieci,
- odbiornik urządzenia pozostaje włączony cały czas,
- jest niezbędnym elementem w procesie dołączania innego urządzenia do sieci Thread.

2. Urządzenie końcowe, ED (ang. *End Device*):
   - nie posiada możliwości przekazywania pakietów do innych węzłów,
   - może komunikować się bezpośrednio tylko z sąsiadującym Ruterem,
   - odbiornik urządzenia może zostać wyłączony w celu ograniczenia zużycia mocy.

ED, stanowiące rolę *Dziecka* (ang. *Child*), pozostaje zawsze połączone z wyłącznie jednym Ruterem, którego określa się terminem *Rodzic* (ang. *Parent* lub *Parent Router*).

1.2.2. Typy urządzeń

Ostatecznie, dokonano podziału urządzeń Thread, wyszczególniając dwa główne typy:

1. FTD (ang. *Full Thread Device*),
2. MTD (ang. *Minimal Thread Device*).

Pełną klasyfikację urządzeń w sieci Thread, z kryterium podziału ze względu na typ, przedstawiono na Rys 1.2.

1.2.2.1. Full Thread Device

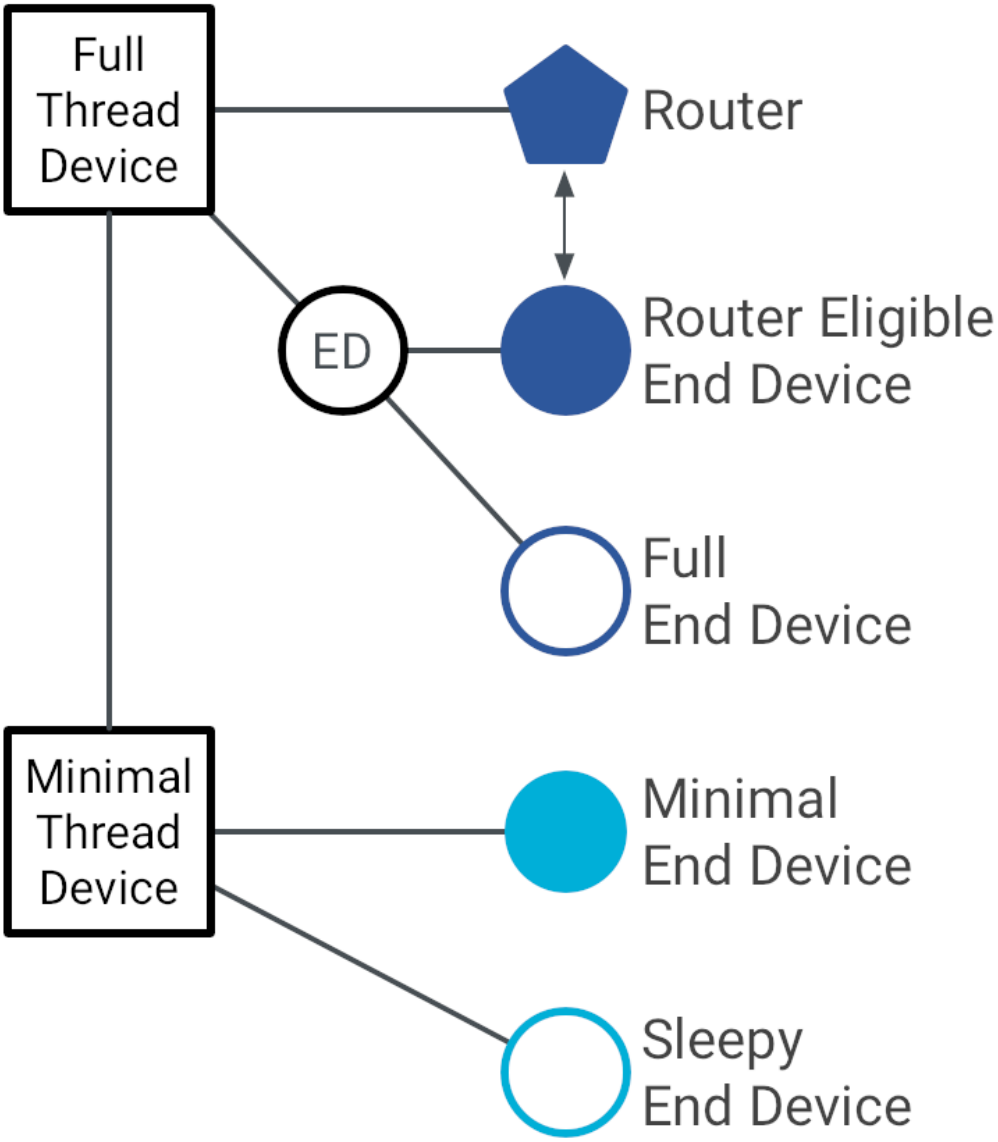
FTD mogą funkcjonować w sieci zarówno w roli Urządzeń Końcowych, jak i w roli Ruterów. Dalej można wyszczególnić:

- Ruter (ang. *Router*) - urządzenie pełniące wspomnianą rolę Rutera.
- REED (ang. *Router Eligible End Device*) - rodzaj Urządzenia Końcowego, które może zostać awansowane do roli Rutera.
- FED (ang. *Full End Device*) - rodzaj Urządzenia Końcowego, które nie może zostać awansowane do roli Rutera.

1.2.2.2. Minimal Thread Device

MTD ograniczone są jedynie do pełnienia roli Urządzeń Końcowych. Kolejno można wyróżnić:

- MED (ang. *Minimal End Device*) - rodzaj Urządzenia Końcowego, którego odbiornik jest zawsze włączony.
- SED (ang. *Sleepy End Device*) - rodzaj Urządzenia Końcowego, którego odbiornik jest wyłączony przez większość czasu, natomiast włącza się okazjonalnie, aby odebrać pakiety przekazane przez Rodzica.



Rys. 1.2. Podział urządzeń w sieci Thread ze względu na typ [4].

1.2.3. Dodatkowe role Rutera

Poza podstawową funkcjonalnością, jaką jest przekazywanie pakietów między węzłami sieci, Ruter w sieci Thread może posiadać dodatkowe role. Między innymi są to:

– Lider (ang. *Leader*),

– Ruter Brzegowy (ang. *Border Router*).

1.2.3.1. Lider

Topologia sieci Thread jest dynamiczna i role poszczególnych węzłów zmieniają się w czasie. Za mechanizm awansu ED do Ruterów oraz degradacji Ruterów do ED odpowiada Ruter pełniący rolę Lidera. Opisane zachowanie jest istotne w momencie wystąpienia awarii, gdy jeden z węzłów nie może pełnić swojej roli.

Taka cecha sieci nazywana jest samoleczeniem (ang. *self-healing*) i jest charakterystyczna dla sieci kratowych Thread.

Główne funkcje Lidera:

– Awans do Ruterów oraz ich degradacja poprzez zarządzanie numerem ID Rutera (ang. *Router ID*),

– Zbierania i rozsyłanie informacji o sieci Thread.

Proces awansu i degradacji węzłów przez Lidera uwzględnia możliwości typów urządzeń opisane w podsekcji 1.2.2. W jednej sieci Thread rolę Lidera może pełnić tylko jedno urządzenia. W momencie tworzenia topologii, urządzenie FTD inicjujące sieć zostaje wybierane na Lidera.

1.2.3.2. Ruter Brzegowy

W celu nawiązania komunikacji między siecią Thread a siecią IP spoza tej domeny, np. Ethernet lub IEEE 802.11, niezbędne jest, aby w sieci przynajmniej jeden Ruter funkcjonował jako Ruter Brzegowy. Posiada on minimum 2 interfejsy sieciowe - jeden interfejs sieciowy Thread oraz drugi zewnętrzny oparty o IP, najczęściej Ethernet lub WLAN (ang. *Wireless Local Area Network*). Zadaniem takiego Routera jest przekazywanie pakietów IPv6 pomiędzy tymi interfejsami.

Wymienione role Rutera mogą się nakładać. Jeden węzeł może być jednocześnie np. Ruterem Brzegowym, Liderem oraz Rodzicem dla innego ED.

1.3. Przegląd wybranych mechanizmów sieci Thread

1.3.1. Tworzenie sieci

Formowanie sieci Thread rozpoczyna się od wybrania numeru kanału (ang. *channel*) oraz identyfikatora PAN ID (ang. *Personal Area Network Identifier*) zgodnie z IEEE 802.15.4. Kolejno, aby stworzyć sieć, urządzenie inicjujące jest zobligowane do wybrania wartości poniższych parametrów:

– *Thread Network Short PAN Identifier (PAN ID)* - 16 bitowa liczba całkowita bez znaku, unikalna w zasięgu skanowania.

– *Network Key* - 16 B, do zastosowań kryptograficznych.

– *Commissioning Credential* - 8-255 B, ciąg znaków, wykorzystywany do tworzenia klucza PSKc podczas procesu Commissioning.

– *Mesh-Local Prefix* - prefix IPv6, wykorzystywany do adresacji urządzeń w sieci Thread.

– *Extended PAN ID* - 8 B, identyfikuje sieć Thread w zasięgu.

– *Network Name* - 1-16 B, ciąg znaków, identyfikuje sieć Thread.

Urządzenie inicjujące sieć zostaje Ruterem, wybierając dla siebie identyfikator *RouterId*, a w konsekwencji braku innych Ruterów w sieci, zostaje również Liderem.

1.3.2. Commissioning

Commissioning to proces pozwalający nowym urządzeniom na dołączenie do istniejącej sieci Thread. W celu zapewnienia bezpieczeństwa oraz powstrzymanie niechcianych urządzeń (ang. *Rouge Device*) przed połączeniem z siecią Thread węzły powinny zostać uwierzytelnione i autoryzowane. Podstawą działania mechanizmu jest protokół DTLS (ang. *Datagram Transport Layer Security*).

Do opisania mechanizmu Commissioning wprowadzono następujące terminy:

- *Joiner* - rola nowego urządzenia próbującego podłączyć się do sieci.
- *Commissioner* - rola urządzenia, który uwierzytelnia węzeł o roli Joiner.
- *Joiner Router* - rola urządzenia, który jest Ruterem i znajduje się najbliżej sieci, do której stara się przyłączyć Joiner.

Ze względu na to czy Commissioner znajduje się w sieci Thread lub poza nią, wyróżniamy kolejno 2 typy procesu Commissioning [5]:

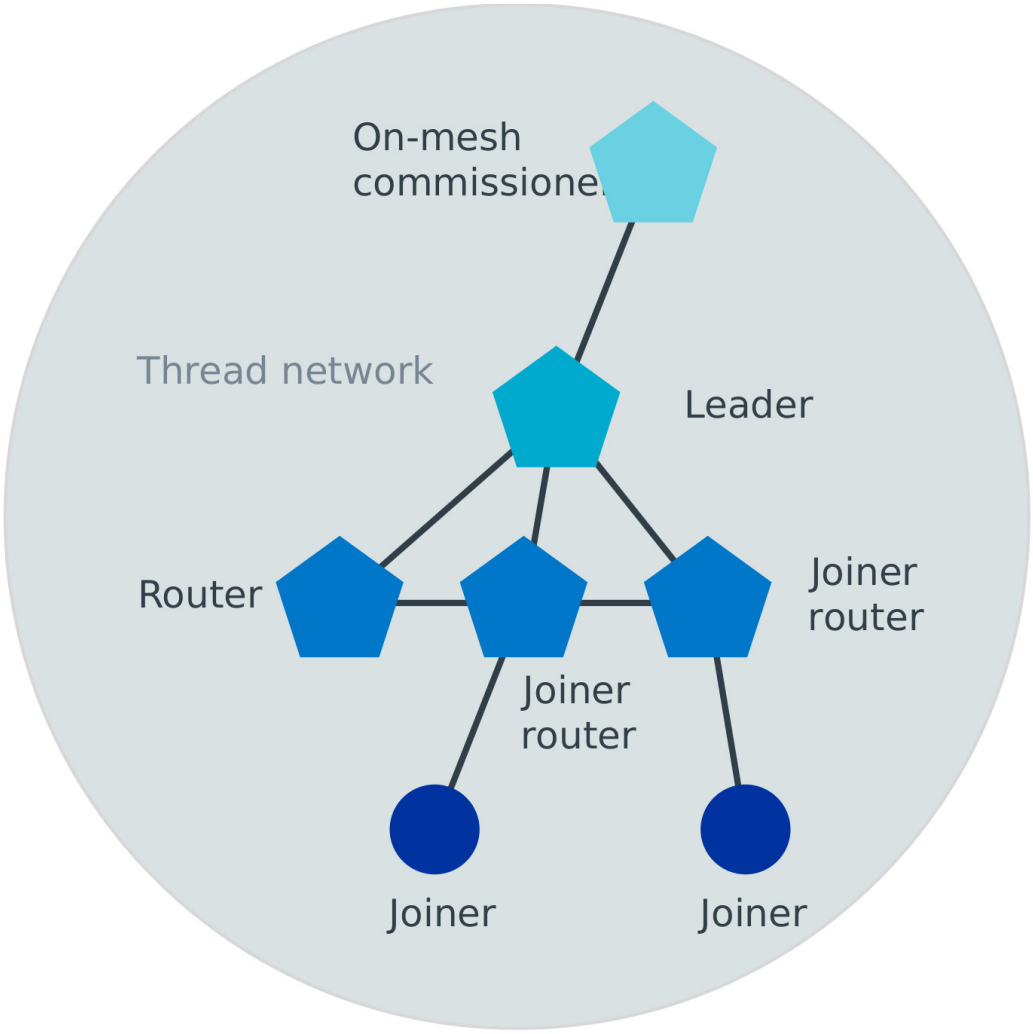
1. *On-mesh*,
2. *External*.

W aktualnej sekcji opisany zostanie jedynie scenariusz On-mesh, w którym Commissioner jest częścią sieci w domenie Thread.

Na Rysunku 1.3 przedstawiono przykładową sieć Thread z wyszczególnieniem ról charakterystycznych dla procesu On-mesh Commissioning.

W sieci Thread, w jednym momencie może istnieć dokładnie jeden Commissioner. Urządzenia, które mogą pełnić rolę Commissioner, wymieniają wiadomości z Liderem, w celu ustalenia, który z kandydatów będzie pełnił tę funkcję. Proces ten określono nazwą *Petitioning*.

Po elekcji urządzenia, który będzie pełnił rolę Commissioner, kolejne urządzenia Joiner są w stanie rozpocząć próbę dołączenia do sieci, nazwaną *Joining*. W kolejnym kroku zestawione zostaje połączenie Joiner-Joiner Router, a następnie Joiner-Commissioner, które jest zabezpieczone z użyciem protokołu DTLS , i do którego zestawienia niezbędne jest podanie przez Joinera Joining Device Credential (PSKd). Następnie Joiner zostaje uwierzytelniony na skutek wymiany kolejnych wiadomości z Commissioner. W wyniku poprawnie przeprowadzonego procesu Commissioning urządzenie posiada wszystkie niezbędne informacje, aby podłączyć się do sieci.



Rys. 1.3. Przykładowa topologia sieci Thread podczas procesu On-mesh Commissioning [5].

1.3.3. Dołączanie do sieci

Proces dołączania do sieci Thread rozpoczyna się od skanowania istniejących sieci IEEE 802.15.4 na kolejnych kanałach. Urządzenie podłączające się rozgłasza żądanie Beacon Request. W przypadku otrzymania wiadomości przez Rutery lub REED odpowiadają one poprzez wysłanie Beacon, który zawiera podstawowe informacje na temat danej sieci.

Po zakończeniu procesu odkrywania sieci Thread urządzenie może próbować podłączyć się do istniejącej sieci lub utworzyć własną.

Aby urządzenie było w stanie dołączyć do sieci Thread, jest zobowiązane do pozyskania poniższych parametrów opisujących konkretną sieć Thread:

- Network Key,
- PSKc,
- Mesh-Local Prefix,

– Extended PAN ID,

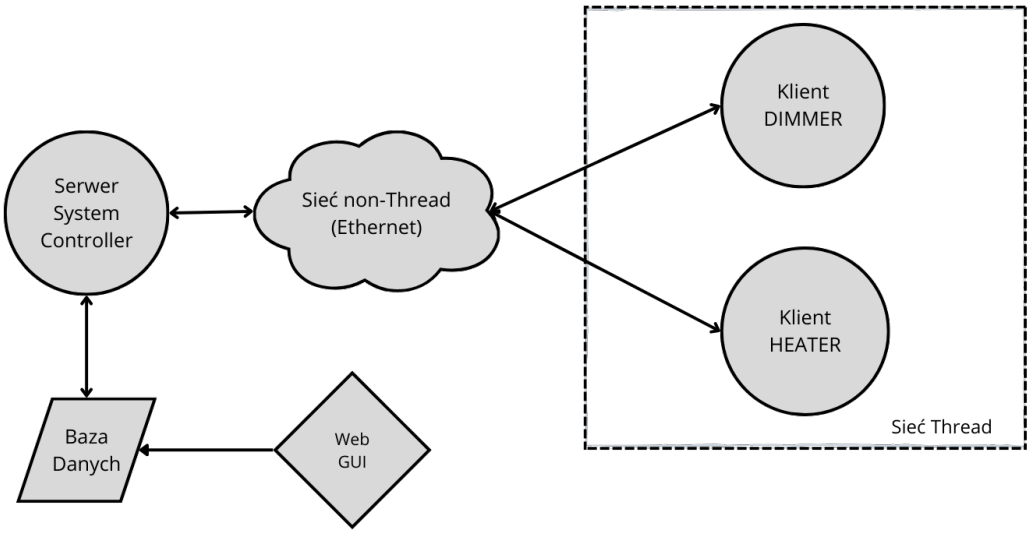
– Network Name.

Informacje te mogą zostać uzyskane w wyniku pomyślnego zakończenia procesu Commissioning.

Urządzenie posiadające wszystkie niezbędne informacje o sieci, w kolejnym kroku zestawia połączenie Dziecko-Rodzic z wykorzystaniem mechanizmu MLE (ang. *Mesh Link Establishment*). Ostatecznie, nawiązanie połączenia zakończone jest przesłaniem przez Rodzica identyfikatora Child ID. Od tego momentu urządzenie dołączające jest częścią sieci Thread.

2. Propozycja systemu

Na Rysunku 2.1 przedstawiono schemat blokowy prototypowego systemu automatyki domowej, którego projekt i implementacja leżą w zakresie kolejnych rozdziałów.



Rys. 2.1. Proponowana architektura systemu automatyki domowej.

W dalszej części pracy termin *parametr regulacji*, będzie odnosił się do procentowego stosunku aktualnej mocy układu regulacji do jego maksymalnej wartości mocy.

Funkcją systemu jest utrzymanie zdefiniowanych w Tabeli 2.1 parametrów środowiska, na poziomie zadanym przez użytkownika systemu. W tym celu urządzenia systemu dokonują pomiaru parametrów oraz regulacji swojej mocy.

Tabela 2.1. Zestaw parametrów określonych dla zdefiniowanych profili.

	HEATER	DIMMER
Parametr	temperatura	natężenie oświetlenia
Jednostka	°C	lux

Zaproponowany system opiera się o architekturę klient-serwer i składa się z 5 funkcjonalnych bloków:

1. klienta Dimmer,

2. klienta Heater,
3. serwera z aplikacją System Controller,
4. Bazy Danych,
5. strony webowej z interfejsem GUI (ang. *Graphical User Interface*).

Celem niniejszego rozdziału jest omówienie zaproponowanego systemu oraz opisanie jego poszczególnych komponentów.

2.1. Strona kliencka

Klientami w systemie są urządzenia połączone w sieć Thread, realizujące profile określone w Tabeli 2.1. Rolą HEATER oraz DIMMER jest implementacja usług pomiaru i regulacji, odpowiednio, temperatury oraz natężenia oświetlenia. Klienci komunikują się z serwerem w celu dostarczenia pomiarów oraz uzyskania informacji o decyzji dotyczącej konieczności regulacji mocy.

2.2. Strona serwera

System Controller to aplikacja znajdująca się w sieci spoza domeny Thread, pełniąca rolę serwera. Komponent przetwarza otrzymane wartości aktualnego stanu środowiska i uwzględniając zadane przez użytkownika wartości, dokonuje decyzji odnośnie regulacji. Ostatecznie aplikacja powiadamia klientów HEATER oraz DIMMER o konieczności zmiany parametru regulacji.

Dodatkowo System Controller odpowiedzialny jest za logowanie otrzymanych wartości aktualnego stanu parametrów oraz podjęte decyzje odnośnie regulacji.

2.3. Sterowanie i logowanie

W celu zadania wartości parametrów, do których powinien dążyć system, użytkownik korzysta z Web GUI. Graficzny interfejs umożliwia zdefiniowanie stanu, jak również wgląd do logów systemu.

Wpisy systemowe oraz zadeklarowany stan parametrów użytkownika, przechowywane są w bazie danych, do której dostęp ma zarówno Web GUI, jak i System Controller.

3. Wykorzystane narzędzia

Celem obecnego rozdziału jest opisanie wykorzystanych do implementacji systemu narzędzi, takich jak platforma sprzętowa, biblioteki programowe, środowisko programistyczne.

3.1. Platforma sprzętowa nRF52833

Do projektu wykorzystano układ Nordic Semiconductor nRF52833 z serii nRF52.

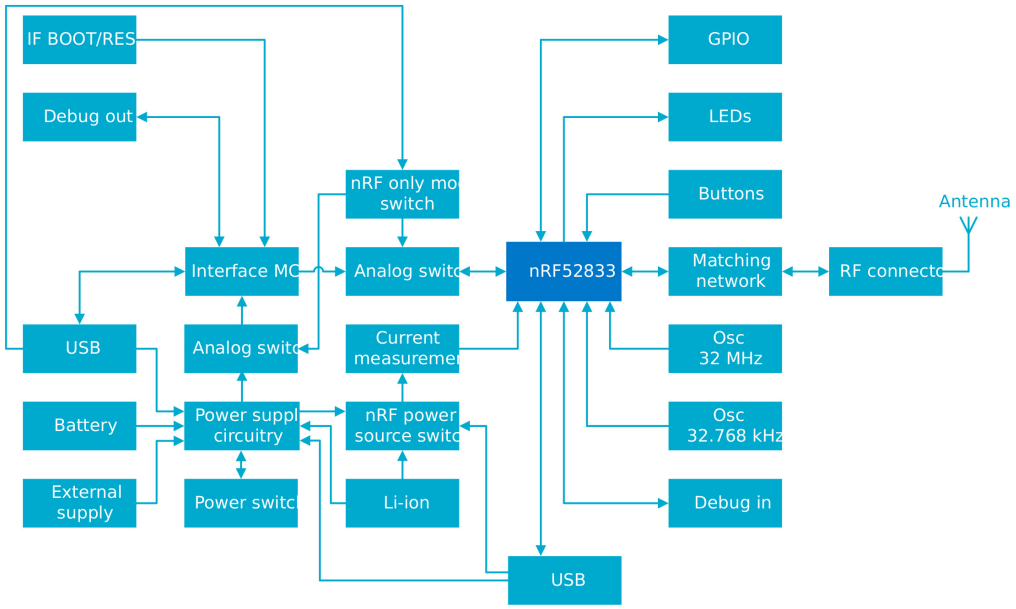
3.1.1. Charakterystyka układu

Platforma nRF52833 to SoC (ang. *System on Chip*) wyposażony w 64 MHz CPU (ang. *Central Processing Unit*) z rdzeniem z rodziny ARM Cortex-M4 oraz jednostką FPU (ang. *Floating Point Unit*) [6]. Układ posiada 512 kB pamięci nieulotnej flash oraz 128 kB pamięci operacyjnej RAM (ang. *Random Access Memory*).

SoC zawiera radiowy układ nadawczo-odbiorczy (ang. *Transreceiver*), umożliwiający odbiór ramek IEEE 802.15.4-2006, co jest kluczowe do uruchomienia aplikacji działającej na stosie Thread.

Według zapewnień producenta platforma nRF52833 może zostać wykorzystana do zastosowań Smart-Home lub przemysłowego IoT i pracować jako czujnik, lub kontroler [6].

Do procesu rozwijania oprogramowania, wykorzystano płytkę ewaluacyjną nRF52833 DK, której schemat blokowy przedstawiono na Rysunku 3.1.



Rys. 3.1. Schemat blokowy płytki ewaluacyjnej nRF52833 DK [7].

3.1.2. Narzędzia producenta

Do tworzenia oprogramowania wykorzystano zestaw narzędzi nRF Connect SDK [8]. Stworzone przez Nordic Semiconductor SDK (ang. *Software Development Kit*) jest dedykowane do tworzenia aplikacji, wykorzystujących transmisję bezprzewodową małej mocy. nRF Connect SDK opiera się między innymi, na urządzeniach z serii nRF52 oraz wspiera rozwój oprogramowania w środowisku Microsoft Windows.

nRF Connect SDK udostępnia system operacyjny czasu rzeczywistego Zephyr oraz przykładowe kody źródłowe aplikacji na nim opartych. Ponadto zestaw narzędzi integruje szereg bibliotek i sterowników z zakresu komunikacji bezprzewodowej, które umożliwiają przyjazne użytkownikowi wdrożenie we własnych aplikacjach, a w szczególności dotyczące technologii:

- Thread,
- CoAP (ang. *Constrained Application Protocol*),
- IPv6,
- UDP,
- IEEE 802.15.4.

W projekcie systemu wykorzystano nRF Connect SDK w wersji 2.4.0.

Podczas tworzenia oprogramowania, użyto również zapewnione przez Nordic Semiconductor rozszerzenie do edytora tekstu Visual Studio Code [9], który umożliwia tworzenie, konfigurowanie i budowanie aplikacji dedykowanych dla platform Nordic Semiconductor oraz debugowanie i programowanie urządzeń z serii nRF52.

3.2. Openthread

OpenThread to upubliczniona, oparta na licencji BSD-3 implementacja stosu Thread, stworzona przez Google. Projekt OpenThread jest wspierany przez, między innymi, Nordic Semiconductor. Stos OpenThread wraz z platformą sprzętową nRF52833 posiada certyfikat *Thread Certified Component* [10], co świadczy o pełnej zgodności ze standardem Thread oraz spełnieniu dodatkowych wymogów wyszczególnionych w specyfikacji [3]. Implementacja OpenThread wraz z API (ang. *Application Programming Interface*) jest uwzględniona w nRF Connect SDK przedstawionym w podsekcji 3.1.2.

Poza implementacją stosu Thread OpenThread dostarcza dodatkowe narzędzia pomocne przy rozwoju systemów opartych o implementację Thread. Jednym z nich jest aplikacja Rutera Brzegowego OTBR (ang. *OpenThread Border Router*) [11].

4. Implementacja systemu

Celem omawianego rozdziału jest przedstawienie implementacji proponowanego w Rozdziale 2 systemu automatyki domowej. Niniejsza część została podzielona na dwie sekcje. W pierwszej sekcji zaprezentowana zostanie stworzona na potrzeby projektu sieć Thread. Kolejny fragment poświęcony jest warstwie aplikacji, technologiom wykorzystanym przy wdrażaniu komponentów oraz zasadzie działania systemu.

Projekt aplikacji systemu znajduje się w stworzonym przez autora pracy repozytorium [12].

4.1. Sieć Thread

4.1.1. Założenia

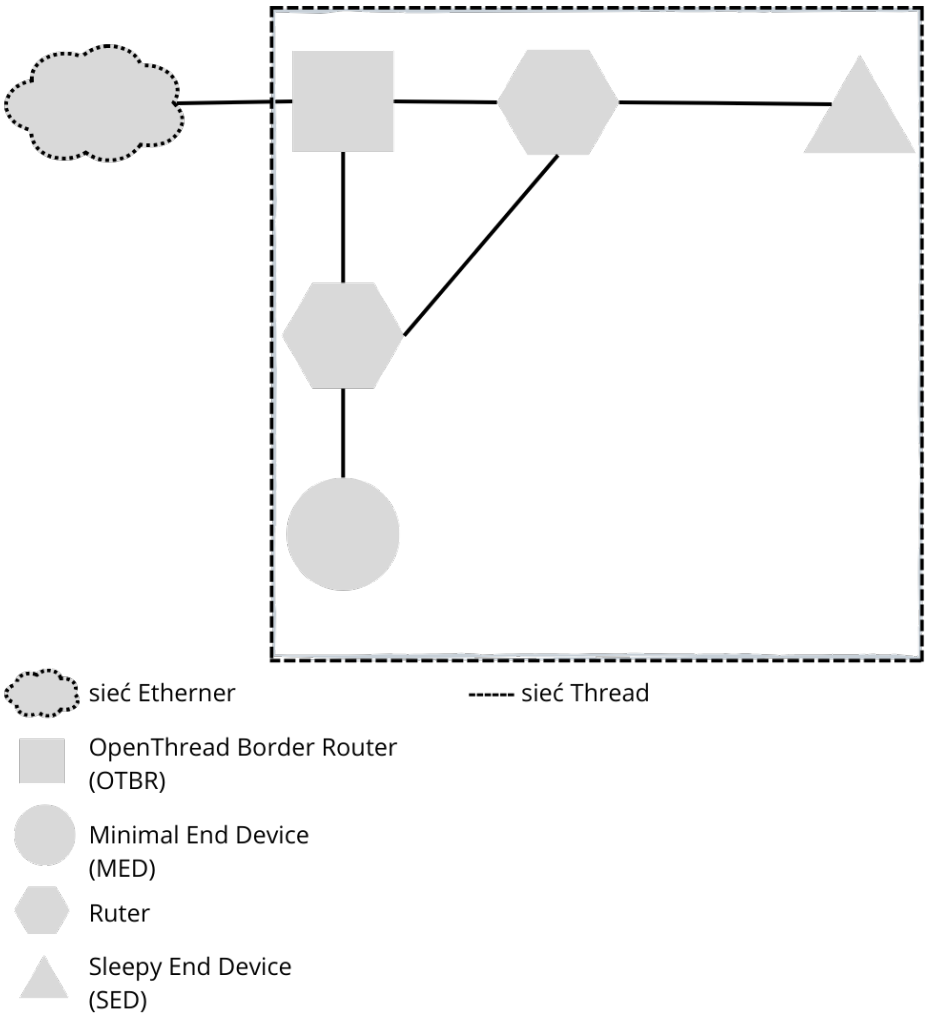
Do stworzenia sieci Thread wykorzystano 5 urządzeń Nordic Semiconductor nRF52833 oraz Laptop Dell G3 15 z procesorem Intel Core i7 9. generacji, z systemem operacyjnym Microsoft Windows 11. Przed przystąpieniem do procesu implementacji sieci Thread, dokonano następujących założeń:

- Jedna platforma nRF52833 wraz z laptopem Dell zostaną wykorzystane do pełnienia roli Rutera brzegowego.

- Dwa urządzenia nRF52833 zostaną skonfigurowane do typu MTD oraz będą stanowiły urządzenia końcowe MED oraz SED, które w warstwie aplikacji będą pełniły rolę profili HEATER oraz DIMMER.

- Dwa urządzenia nRF52833 zostaną skonfigurowane do typu FTD. Zostaną one wprowadzone do sieci, aby zapewnić nadmiarowość i pozwolić Liderowi Thread na dostosowanie topologii do aktualnych potrzeb i wymagań sieci.

Na Rysunku 4.1 przedstawiono przykładową topologię sieci Thread, złożoną z wyszczególnionych powyżej urządzeń.



Rys. 4.1. Przykładowa topologia sieci złożonej z urządzeń wymienionych w Sekcji 4.1.1.

4.1.2. Ruter Brzegowy

W celu zapewnienia komunikacji przyszłej sieci Thread z zewnętrzną siecią Ethernet, w pierwszej kolejności przystąpiono do konfiguracji Rutera brzegowego, wykorzystując implementację OTBR.

Aplikacja OTBR jest dedykowana dla systemów operacyjnych Linux oraz do poprawnego działania wymaga nawiązania komunikacji zarówno z interfejsem sieci Thread, jak i z interfejsem sieci zewnętrznej. Z tego powodu, w początkowym kroku skonfigurowano platformę Linux jako maszynę wirtualną z dystrybucją Ubuntu tak, aby zapewnić komunikację sieciową oraz komunikację portów szeregowych między urządzeniem gościa (ang. *Guest*) (maszyna wirtualna) a gospodarza (ang. *Host*) (Laptop Dell). Kolejno skonfigurowano środowisko deweloperskie, instalując narzędzia takie jak system kontroli wersji Git oraz IDE (ang. *Integrated Development Environment*).

Po weryfikacji poprawności działania maszyny wirtualnej kontynuowano proces konfiguracji OTBR, wykorzystując instrukcję zamieszczoną na oficjalnej stronie projektu OpenThread [13]. Zaprogramowano jedno z urządzeń nRF52833 aplikacją RCP (ang. *Radio Co-Processor*), używając dostarczonej przez OpenThread

implementacji [14]. Ostatecznie zainstalowano oprogramowanie Docker oraz pobrano dystrybuowany przez OpenThread kontener.

Tak przygotowane środowisko Linux oraz zaprogramowane urządzenie nRF52833 są gotowe do uruchomienia w pełni funkcjonalnej aplikacji Rutera Brzegowego.

4.1.3. Urządzenia MTD oraz FTD

Aplikację dla 4 pozostałych urządzeń nRF52833 stworzono w języku C z wykorzystaniem nRF Connect SDK oraz środowiska programistycznego nRF Connect IDE. W plikach konfiguracyjnych projektów skonfigurowano odpowiednio stos protokołu i niezbędne funkcjonalności Thread, system logowanie, GPIO (ang. *general-purpose input/output*). Kolejno wybrano tryb MTD dla 2 urządzeń, natomiast dla pozostałych FTD.

Co więcej, w urządzeniach MTD uwzględniono możliwość przejścia w tryb energooszczędny, poprzez naciśnięci przycisku Button 3, w wyniku czego ED zmienia pełniącą rolę z MED na SED oraz obniża zużycie energii w wyniku ograniczenia zużycia pamięci RAM.

Na potrzeby procesu rozwoju oprogramowania, wzbogacono wszystkie 4 platformy o sygnalizację stanu urządzenia Thread przy pomocy LED (ang. *light-emitting diode*). Dioda LED1 świeci się, gdy urządzenie Thread jest włączone do sieci.

4.2. Warstwa Aplikacji

Do wymiany informacji podczas komunikacji klient-serwer między urządzeniami końcowymi w sieci Thread a System Controllerem, wykorzystano protokół warstwy aplikacji Constrained Application. CoAP jest protokołem opartym o REST API (ang. *Representational State Transfer Application Programming Interface*), który nawiązuje połączenie z wykorzystaniem wspieranego przez stos Thread protokołu UDP.

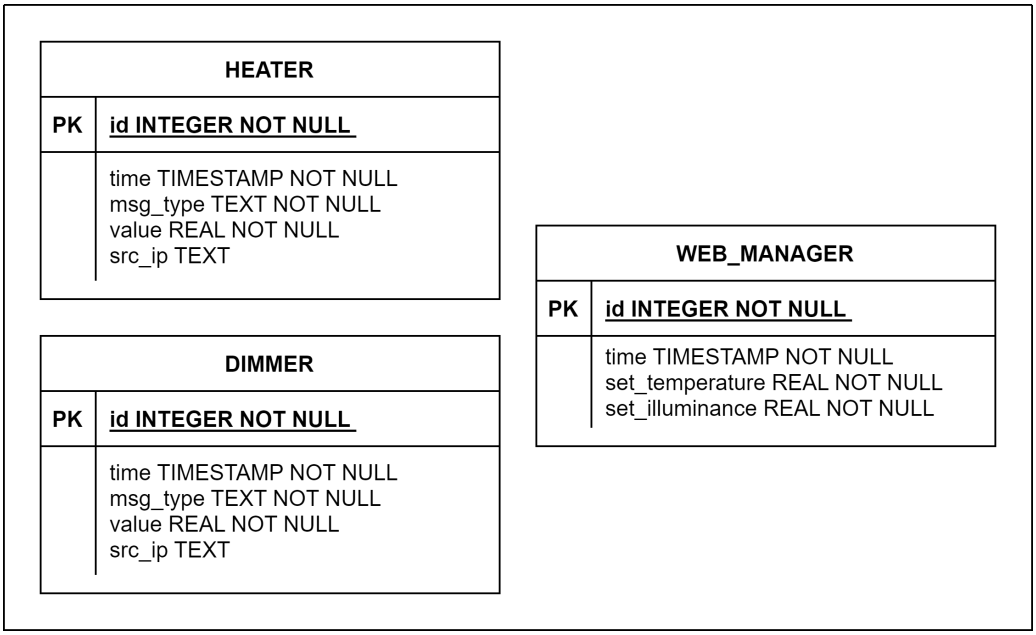
4.2.1. Implementacja komponentów systemu

Niniejsza sekcja ma za zadanie przybliżyć szczegóły implementacji poszczególnych komponentów wymienionych w Rozdziale 2.

4.2.1.1. Baza danych

Przez wzgląd na mały rozmiar, pełne wsparcie funkcjonalności SQL (ang. *Structured Query Language*) oraz możliwość przechowywania bazy danych w pojedynczym pliku, jako silnik wykorzystywanej w projekcie systemu Bazy Danych wybrano SQLite. [15].

Stworzony na potrzeby systemu zestaw tabel Bazy Danych przedstawiono na Rysunku 4.2.



Rys. 4.2. Schemat tabel Bazy Danych systemu.

Tabela WEB_MANAGER ma za zadanie przechowywać wpisy dotyczące ustawionych przez użytkownika parametrów. Tablica HEATER oraz tablica DIMMER zawierają informację o kolejnych przetworzonych przez System Controller żądaniach CoAP.

4.2.1.2. System Controller

System Controller został wdrożony jako aplikacja w języku Python, przeznaczona dla urządzeń z systemem operacyjnym Linux. System Controller odbiera żądania oraz wysyła odpowiedzi CoAP z wykorzystaniem dostarczonego przez bibliotekę aiocoap API [16]. URI (ang. *Uniform Resource Identifier*) zdefiniowanych zasobów serwera, obsługiwany typ zapytań oraz ich funkcje, zestawiono w Tabeli 4.1.

Zarządzanie Bazą Danych SQLite odbywa się z użyciem wbudowanych w język Python bibliotek *sqlite3* oraz *aiosqlite*.

Tabela 4.1. Zasoby serwera CoAP.

Zasób	Typ zapytań	Funkcja
temperature	GET, PUT	Przechowuje wartość aktualnej temperatury środowiska.
illuminance	GET, PUT	Przechowuje wartość aktualnego natężenia oświetlenia środowiska.
heater_regulation	GET	Zwraca wartość parametru regulacji dla układu HEATER.
dimmer_regulation	GET	Zwraca wartość parametru regulacji dla układu DIMMER.

4.2.1.3. HEATER oraz DIMMER

Platformy nRF52833 pracujące jako MTD, w których skonfigurowano stos Thread, jak opisano w Podsekcji 4.1.3, wzbogacono o warstwę aplikacji. Urządzenia Końcowe symulują zachowanie zdefiniowanych w Sekcji

2.1 profili HEATER oraz DIMMER. Wysyłanie zapytań oraz odbieranie odpowiedzi CoAP zaimplementowano z wykorzystaniem API dostarczonego przez nRF Connect SDK, oraz OpenThread.

W urządzeniach HEATER oraz DIMMER skonfigurowano przycisk Button 3, którego naciśnięcie inicjalizuje stronę klienta CoAP oraz pobiera z sieci prefiks NAT64, niezbędny do nawiązania połączenia z serwerem o skonfigurowanym IPv4.

4.2.1.4. Web GUI

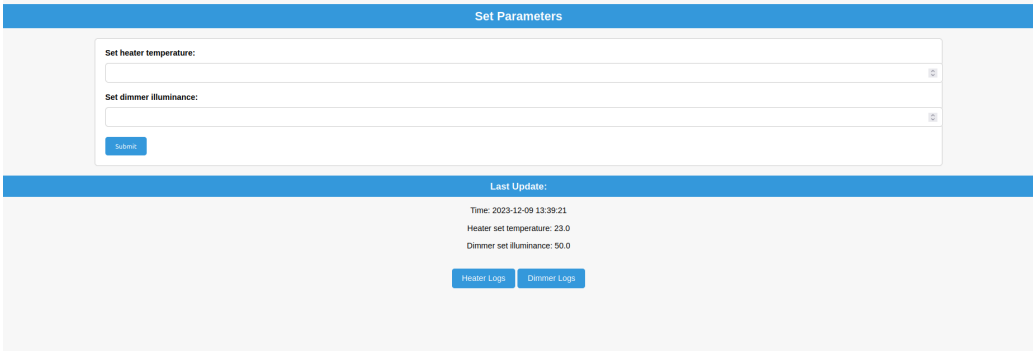
Interfejs użytkownika Web GUI stanowi aplikacja w języku Python, którą stworzono z użyciem następujących technologii webowych:

– mikro-frameworku Flask [17],

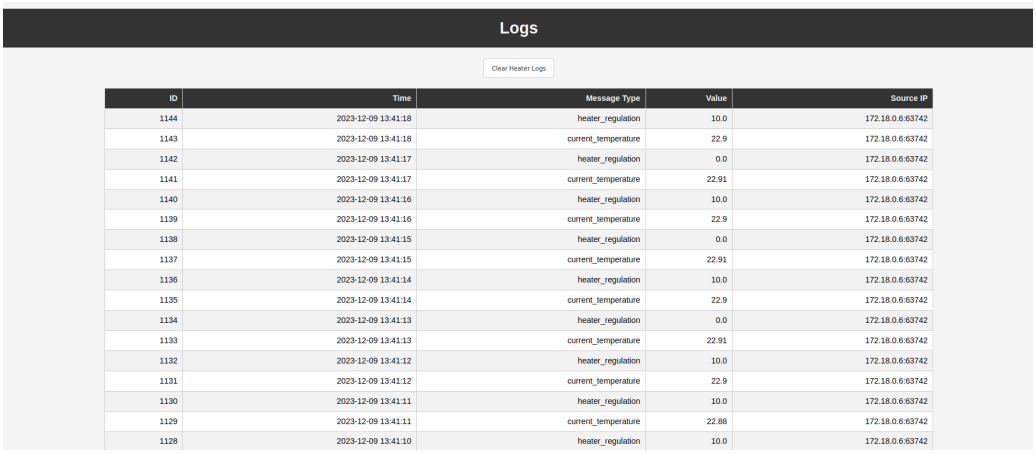
– CSS (ang. *Cascading Style Sheets*),

– HTML (ang. *HyperText Markup Language*).

Aplikacja komunikuje się bezpośrednio z bazą danych, wykorzystując bibliotekę sqlite3.



Rys. 4.3. Zrzut ekranu z panelem Web GUI przeznaczonym do ustalania parametrów systemu.



ID	Time	Message Type	Value	Source IP
1144	2023-12-09 13:41:18	heater_regulation	10.0	172.18.0.6:63742
1143	2023-12-09 13:41:18	current_temperature	22.9	172.18.0.6:63742
1142	2023-12-09 13:41:17	heater_regulation	0.0	172.18.0.6:63742
1141	2023-12-09 13:41:17	current_temperature	22.91	172.18.0.6:63742
1140	2023-12-09 13:41:16	heater_regulation	10.0	172.18.0.6:63742
1139	2023-12-09 13:41:16	current_temperature	22.9	172.18.0.6:63742
1138	2023-12-09 13:41:15	heater_regulation	0.0	172.18.0.6:63742
1137	2023-12-09 13:41:15	current_temperature	22.91	172.18.0.6:63742
1136	2023-12-09 13:41:14	heater_regulation	10.0	172.18.0.6:63742
1135	2023-12-09 13:41:14	current_temperature	22.9	172.18.0.6:63742
1134	2023-12-09 13:41:13	heater_regulation	0.0	172.18.0.6:63742
1133	2023-12-09 13:41:13	current_temperature	22.91	172.18.0.6:63742
1132	2023-12-09 13:41:12	heater_regulation	10.0	172.18.0.6:63742
1131	2023-12-09 13:41:12	current_temperature	22.9	172.18.0.6:63742
1130	2023-12-09 13:41:11	heater_regulation	10.0	172.18.0.6:63742
1129	2023-12-09 13:41:11	current_temperature	22.88	172.18.0.6:63742
1128	2023-12-09 13:41:10	heater_regulation	10.0	172.18.0.6:63742

Rys. 4.4. Zrzut ekranu z panelem Web GUI przeznaczonym do obserwacji logów systemu.

4.2.2. Symulacja

W wyniku założenia o symulacyjnym charakterze prototypowego systemu tj. braku uwzględnienia rzeczywistych czujników oraz układów regulacyjnych zaimplementowano ekosystem, który stanowi pewne przybliżenie środowiska, w którym mógłby operować system.

W układach HEATER oraz DIMMER zasymulowano zmiany temperatury, oraz natężenia oświetlenia z wykorzystaniem licznika systemowego. Wraz z przerwaniem licznika, które dla HEATER następuje co 100ms, natomiast dla DIMMER co 200ms, aktualna temperatura oraz aktualne natężenie oświetlenia zastępowane są nową temperaturą, oraz nowym natężeniem oświetlenia.

Temperatura środowiska w układzie HEATER opisana jest następującym wzorem:

$$nowa\_temperatura = aktualna\_temperatura + a \cdot parametr\_regulacji + b$$

Gdzie:

- *a = 0,00025*,
- *b = -5a*.

Natomiast zmiany natężenia środowiska w układzie DIMMER opisuje funkcja:

$$nowe\_no = aktualne\_no + a \cdot parametr\_regulacji$$

Gdzie:

- *nowe_no* - nowe natężenie oświetlenia,
- *aktualne_no* - aktualne natężenie oświetlenia,
- *a = 0,00025*

Wprowadzony parametr b we wzorze na chwilową temperaturę ekosystemu w układzie HEATER pozwala na zasymulowanie powolnego spadku temperatury. Środowisko wdrożone w Układzie DIMMER można przyjąć za izolowane, ponieważ nie uwzględnia żadnych zewnętrznych źródeł światła. Parametry a oraz b dla obydwu układów, zostały wyznaczone empirycznie, aby zagwarantować możliwość zaobserwowania zmian wynikających z poprawnego działania systemu.

4.2.3. Zasada działania

Celem niniejszej podsekcji jest opisanie zasady działania wdrożonego systemu automatyki domowej, poprzez przegląd logiki aplikacji oraz przepływu informacji między komponentami.

Zachowanie systemu można podzielić na 3 fazy:

- Fazę ustalania parametrów systemu - użytkownik nadaje systemowi stan, do którego mają dążyć układy regulacyjne.

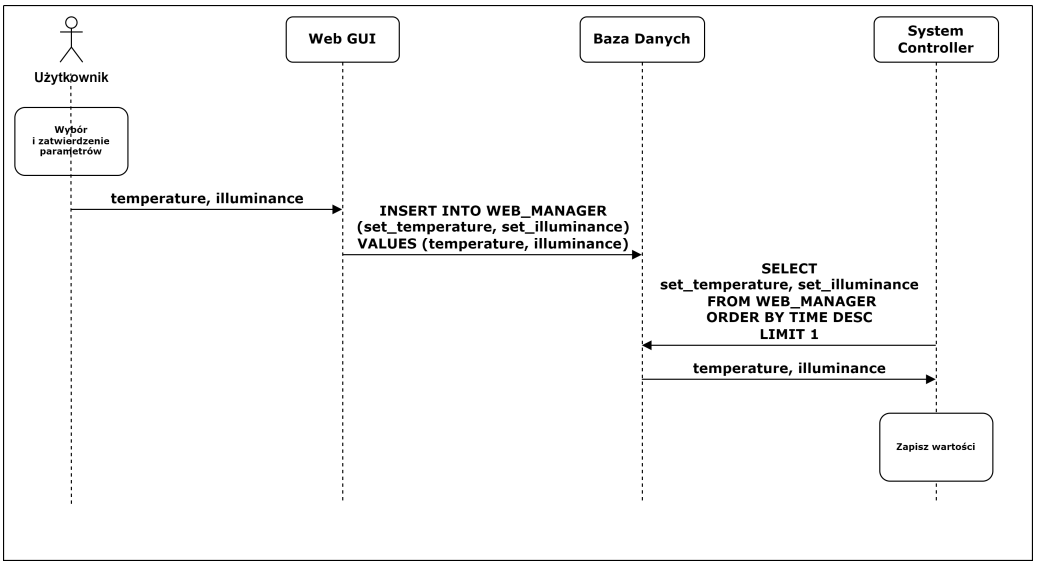
– Fazę pomiarów - układ pomiarowy przesyła aktualny stan parametru System Controllerowi.

– Fazę regulacji - układ regulacji odpytuje System Controller o nową wartość parametru regulacji w celu utrzymania parametrów środowiska dążących do zadanego przez użytkownika stanu.

Ustalanie parametrów systemu odbywa się asynchronicznie przez użytkownika, natomiast regulacja oraz pomiar wykonywane są periodycznie z okresem 1s.

4.2.3.1. Ustalanie parametrów systemu

Na Rysunku 4.5 zilustrowano przepływ wiadomości między komponentami systemu w fazie ustalania parametrów systemu.



Rys. 4.5. Diagram sekwencji ustawiania parametrów systemu.

Kolejnymi krokami procedury są:

1. Użytkownik korzystający z panelu Web GUI, służącego do ustawiania parametrów systemu, zobrazowanym na Rysunku 4.3, podaje i zatwierdza wartości.

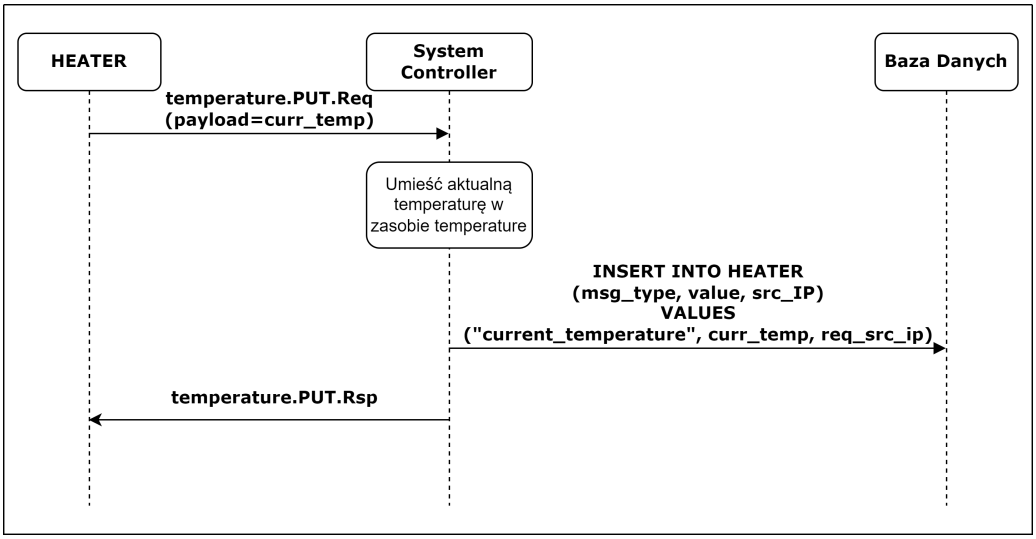
2. Wprowadzone wartości *temperature* oraz *illuminance* wstawiane są do tabeli WEB_MANAGER Bazy Danych.

3. System Controller cyklicznie odpytuje Bazę Danych o ostatnio zaktualizowane wartości temperatury i natężenia oświetlenia.

4. Po uzyskaniu wartości *temperature* oraz *illuminance* System Controller zapisuje je w swoim programie.

4.2.3.2. Pomiar

Na Rysunku 4.6 zilustrowano przepływ wiadomości między komponentami systemu w fazie pomiaru temperatury.

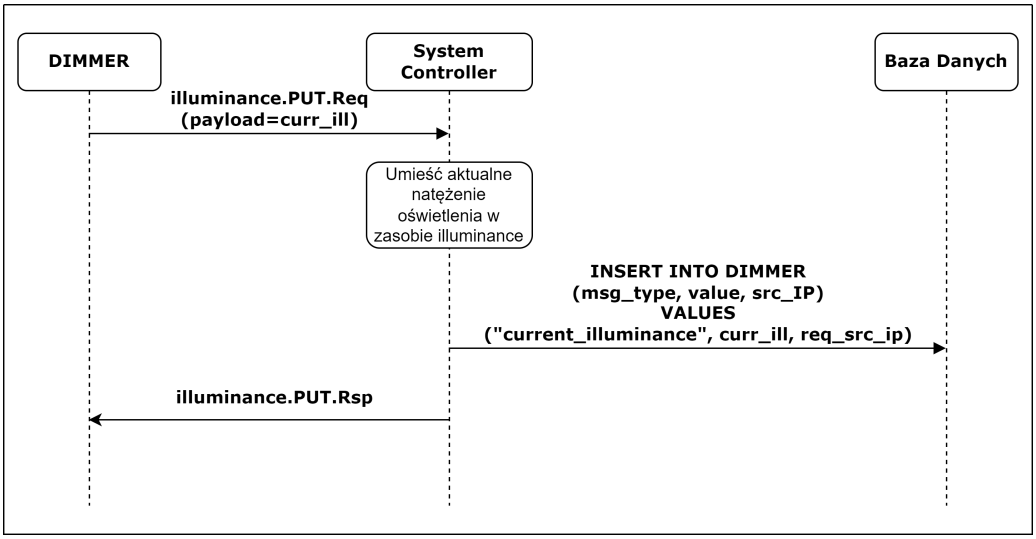


Rys. 4.6. Diagram sekwencji pomiaru temperatury.

Kolejnymi krokami pomiaru temperatury w fazie pomiaru są:

1. Urządzenie HEATER wysyła zapytanie do System Controllera o umieszczenie aktualnej temperatury *curr_temp* w zasobie *temperature*.
2. System Controller umieszcza otrzymaną wartość *curr_temp* w zasobach serwera.
3. System Controller loguje informację o przetworzonym zapytaniu, umieszczając otrzymaną wartość temperatury w tabeli HEATER Bazy Danych.
4. System Controller w ramach potwierdzenia otrzymania zapytania odpowiada układowi HEATER.

Na Rysunku 4.7 zilustrowano przepływ wiadomości między komponentami systemu w fazie pomiaru natężenia oświetlenia.



Rys. 4.7. Diagram sekwencji pomiaru natężenia oświetlenia.

Kolejnymi krokami pomiaru natężenia oświetlenia w fazie pomiaru są:

1. Urządzenie DIMMER wysyła zapytanie do System Controllera o umieszczenie aktualnego natężenia oświetlenia *curr_ill* w zasobie *illuminance*.

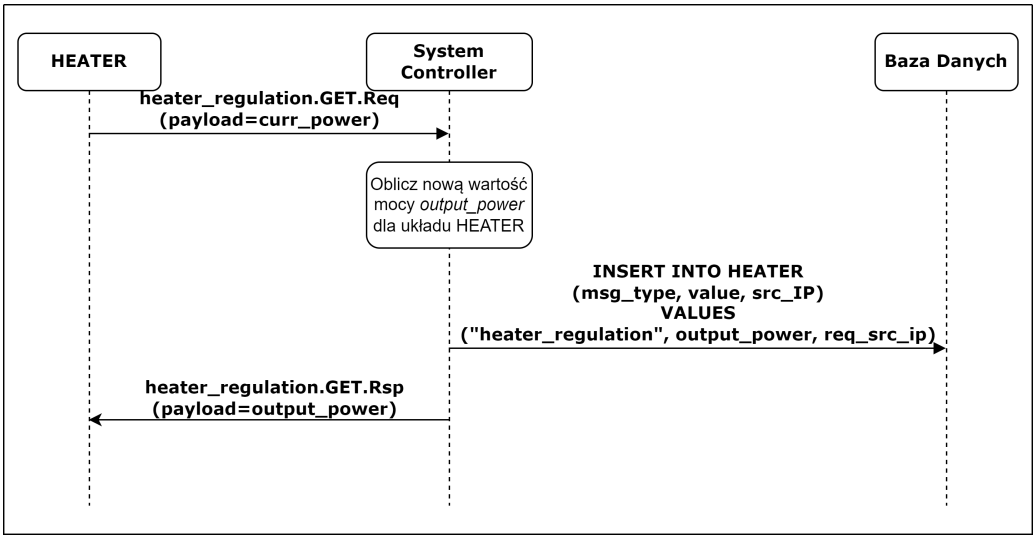
2. System Controller umieszcza otrzymaną wartość natężenia oświetlenia *curr_ill* w zasobach serwera.

3. System Controller loguje informację o przetworzonym zapytaniu, umieszczając otrzymaną wartość natężenia oświetlenia w tabeli DIMMER Bazy Danych.

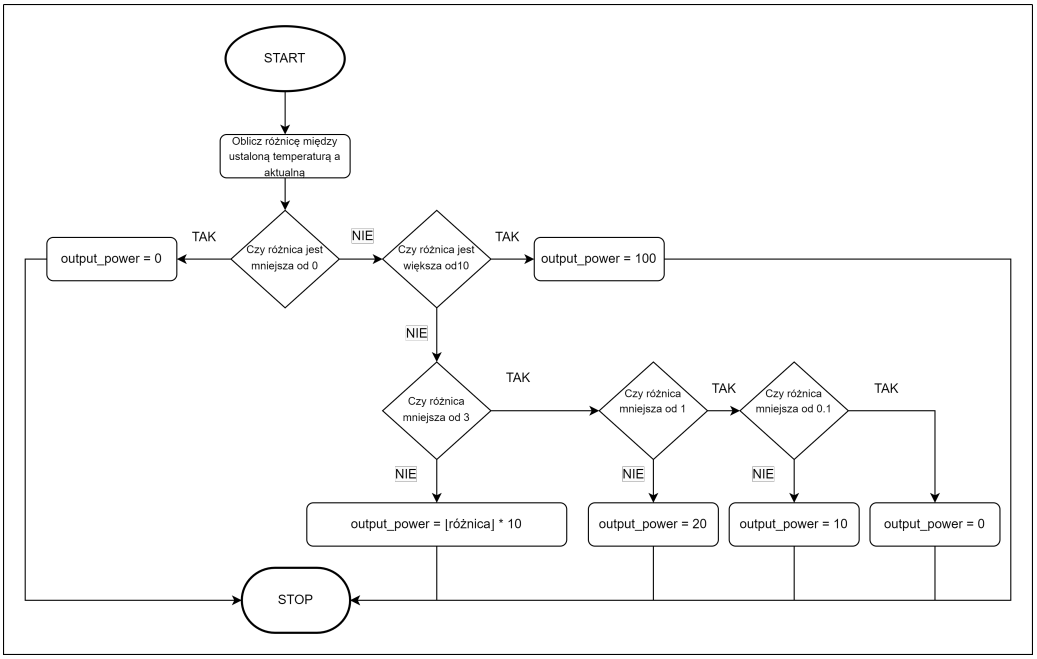
4. System Controller w ramach potwierdzenia otrzymania zapytania odpowiada układowy HEATER.

4.2.3.3. Regulacja

Na Rysunku 4.8 zilustrowano przepływ wiadomości między komponentami systemu w fazie regulacji temperatury.



Rys. 4.8. Diagram sekwencji regulacji temperatury.



Rys. 4.9. Algorytm obliczania parametru mocy dla układu HEATER.

Kolejnymi krokami regulacji temperatury w fazie regulacji są:

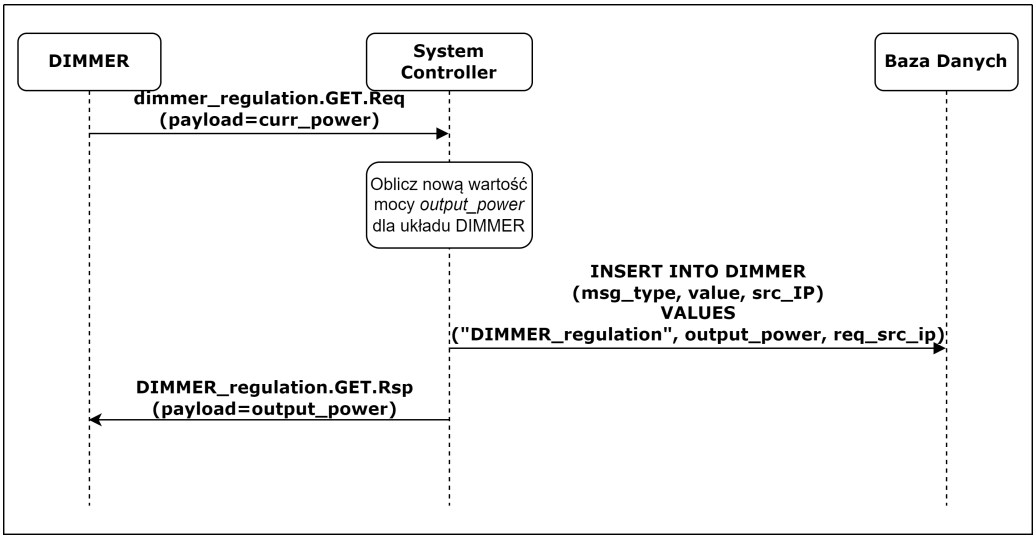
1. Urządzenie HEATER wysyła zapytanie do System Controllera o otrzymanie nowego parametru regulacji w zasobie *heater_regulation*, w którym umieszcza aktualną wartość parametru regulacji *curr_power*.

2. System Controller oblicza nową wartość *output_power* na podstawie aktualnej temperatury, umieszczonej w zasobie *temperature* oraz aktualnej wartości parametru regulacji *curr_power*, zgodnie z algorytmem przedstawionym na Rysunku 4.9.

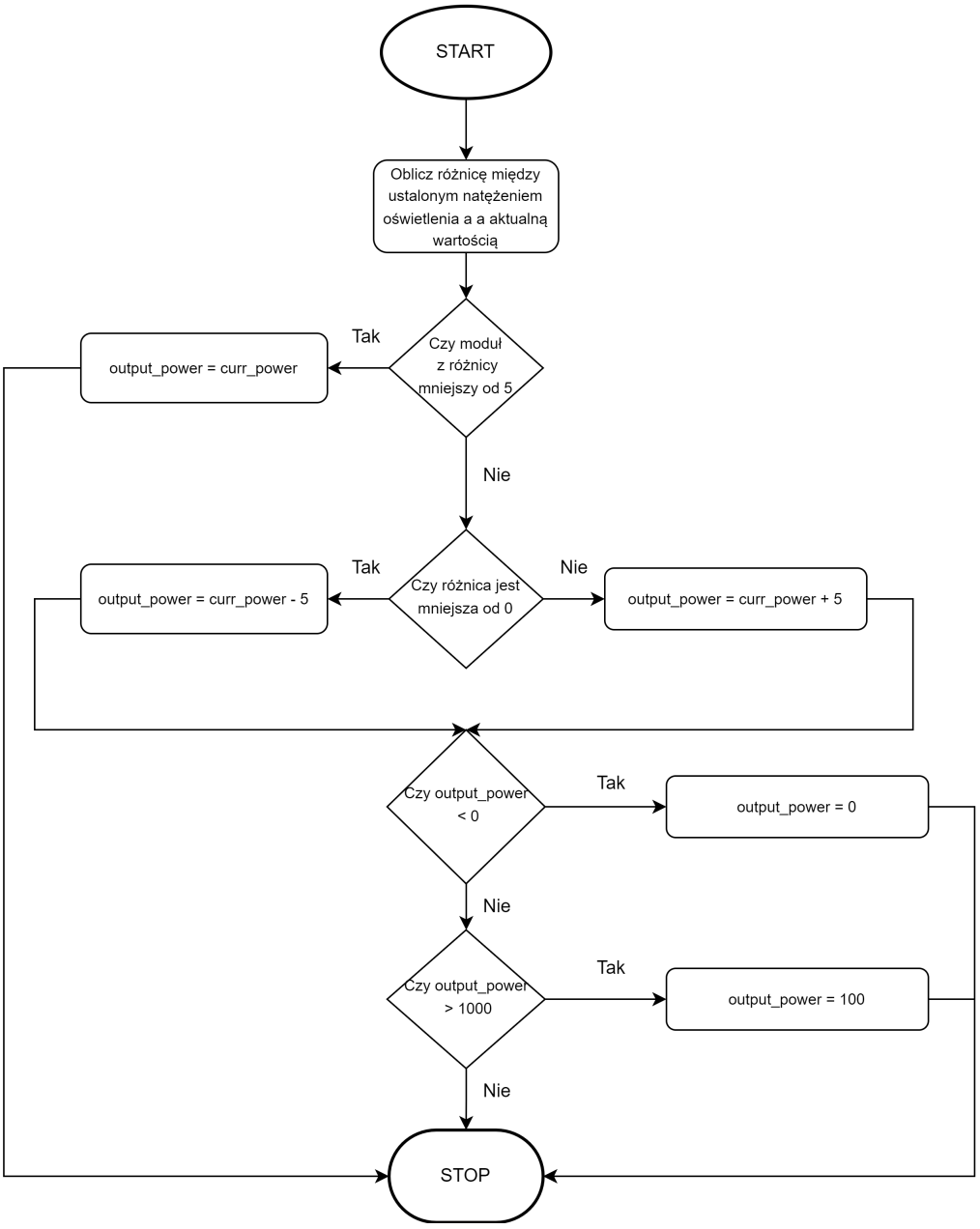
3. System Controller loguje informację o przetworzonym zapytaniu, umieszczając obliczoną wartość *output_power* w tabeli HEATER Bazy Danych.

4. System Controller odpowiada układowy HEATER, dostarczając nową wartość parametru regulacji *output_power*.

Na Rysunku 4.10 zilustrowano przepływ wiadomości między komponentami systemu w fazie regulacji natężenia oświetlenia.



Rys. 4.10. Diagram sekwencji regulacji natężenia oświetlenia.



Rys. 4.11. Algorytm obliczania parametru regulacji dla układu DIMMER.

Kolejnymi krokami regulacji natężenia oświetlenia w fazie regulacji są:

1. Urządzenie DIMMER wysyła zapytanie do System Controllera o otrzymanie nowego parametru regulacji w zasobie *dimmer_regulation*, w którym umieszcza aktualną wartość parametru regulacji *curr_power*.
2. System Controller oblicza nową wartość *output_power* na podstawie wartości aktualnego natężenia oświetlenia, umieszczonego w zasobie *illuminance* oraz poprzedniej wartości parametru regulacji *curr_power*, zgodnie z algorytmem przedstawionym na Rysunku 4.11.
3. System Controller loguje informację o przetworzonym zapytaniu, umieszczając obliczoną wartość *output_power* w tabeli DIMMER Bazy Danych.

4. System Controller odpowiada układowy DIMMER, dostarczając nową wartość parametru regulacji *output_power*.

W związku z charakterem pracy, w której nacisk implementacji projektu został położony na aspekty komunikacji, dobrane algorytmy przedstawione na Rysunku 4.9 oraz Rysunku 4.11 pozwalają na zaprezentowanie działania prototypowego systemu, natomiast nie są optymalne. We wdrożonych algorytmach System Controllera brakuje histerezy, co w konsekwencji mogłoby spowodować skokowe zmiany obliczanego parametru regulacji przy niewielkich odchyleniach wartości temperatury oraz natężenie oświetlenia środowiska w stosunku do zadanego stanu. Opisane zachowanie mogłoby mieć wpływ na żywotność baterii urządzeń, które pracują jako regulator. Jednym z rozwiązań problemu mogłoby być zastosowanie w System Controllerze algorytmu opartego o PID (ang. *Proportional-Integral-Derivative*).

5. Uruchomienie i weryfikacja poprawności działania systemu

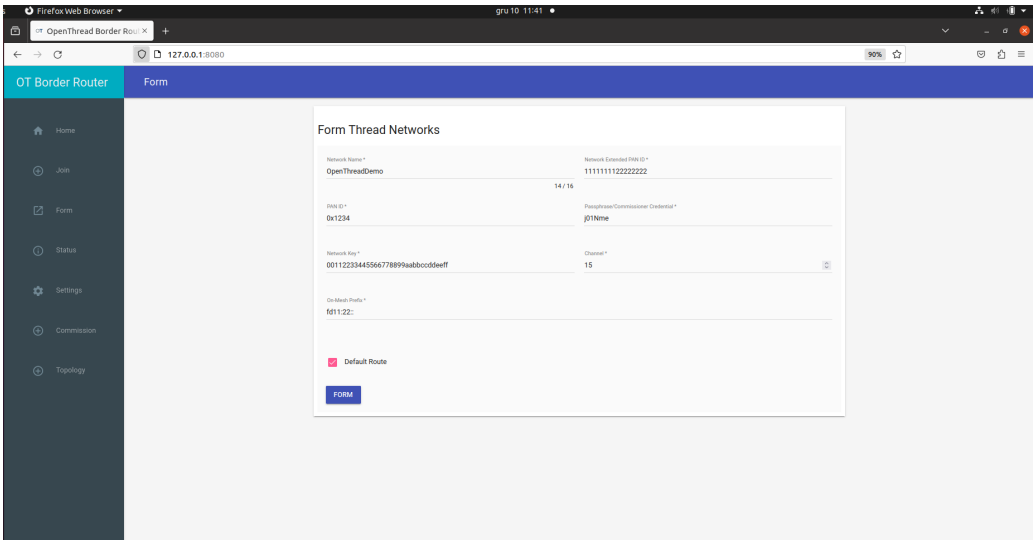
Celem niniejszego rozdziału jest opisanie procesu uruchomienia systemu i weryfikacji poprawności działania, poprzez sprawdzenie połączeń między węzłami sieci Thread oraz zasad działania systemu, przedstawionych w podsekcji 4.2.3.

5.1. Uruchomienie

Pierwszym krokiem w uruchomieniu systemu jest utworzenie sieci Thread. Proces tworzenie topologii rozpoczęto od wystartowania aplikacji OTBR.

Podłączono zaprogramowaną platformę, jak omówiono w Podsekcji 4.1.2, do portu szeregowego urządzenia gospodarza, a następnie sprawdzono poprawność komunikacji między maszyną wirtualną a urządzeniem pracującym jako RCP. Ostatecznie uruchomiono kontener Docker, zawierający aplikację OTBR, wykorzystując przygotowany wcześniej skrypt *my-run-otbr.sh* znajdujący się w repozytorium [12]. Upewniwszy się o poprawnym działaniu Rutera Brzegowego, rozpoczęto tworzenie sieci Thread.

Do konfiguracji sieci wykorzystano interfejs graficzny OTBR Web GUI, będący serwisem zapewnionym w ramach aplikacji OTBR. Zrzut ekrany przedstawiającego wymieniony interfejs graficzny, zaprezentowano na Rysunku 5.1.



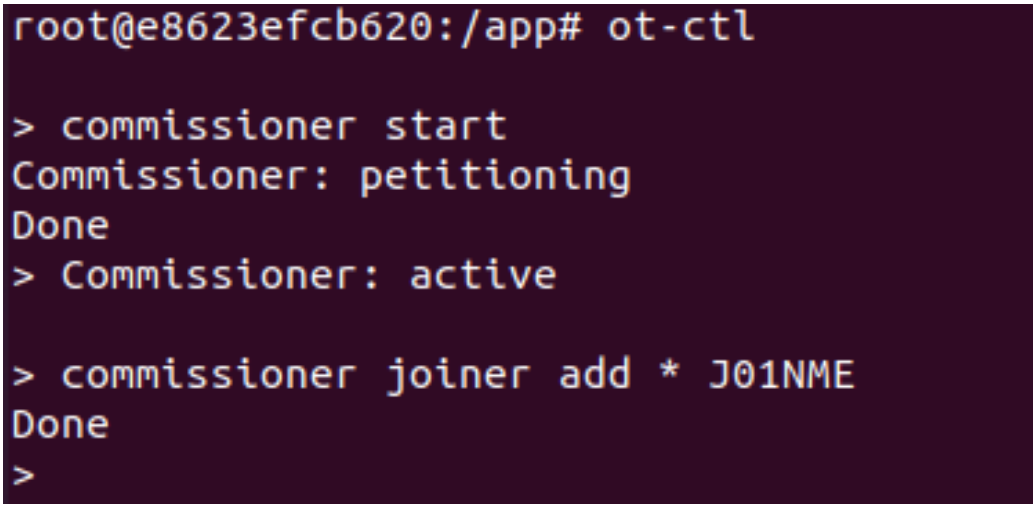
Rys. 5.1. Zrzut ekranu interfejsu OTBR GUI z oknem do tworzenia sieci Thread.

Wartości parametrów koniecznych do stworzenia sieci Thread, które opisano w Podsekcji 1.3.1, przedstawiono w Tabeli 5.1.

Tabela 5.1. Parametry sieci Thread.

Nazwa parametru	Wartość
Network Name	pawel_network
PAN ID	0x0457
Network Key	00112233445566778899aabbccddeeff
Extended PAN ID	fb020000abcd0000
Commissioning Credential	J01NME
Channel	13
Mesh-Local Prefix	fd11:22::

Po zatwierdzeniu parametrów, a w konsekwencji utworzeniu sieci Thread, skonfigurowano Ruter brzegowy, aby pracował jako Commissioner w procesie przedstawionym w Podsekcji 1.3.2. W tym celu wykorzystano OpenThread CLI [18]. Narzędzie to pozwala na konfigurację urządzeń sieci Thread, w których wdrożono OpenThread. Na Rysunku 5.2 przedstawiono wprowadzone w konsolę OpenThread CLI komendy konfigurujące rolę Commissioner w Ruterze brzegowym oraz pozwalające na dołączenie do sieci wszystkim urządzeniom posiadającym Commissioning Credential o wartości *J01NME*.



Rys. 5.2. Zrzut ekranu konsoli OpenThread CLI, przedstawiający konfigurację roli Commissioner.

Tak skonfigurowane urządzenie jest w pełni funkcjonalnym Ruterem brzegowym, pracującym w roli Commissioner.

W kolejnym kroku uruchomiono pozostałe platformy nRF52833 pracujące jak urządzenia FTD oraz HEATER i DIMMER. Po włączeniu urządzeń przystąpiono do konfiguracji roli Joiner każdego z węzłów.

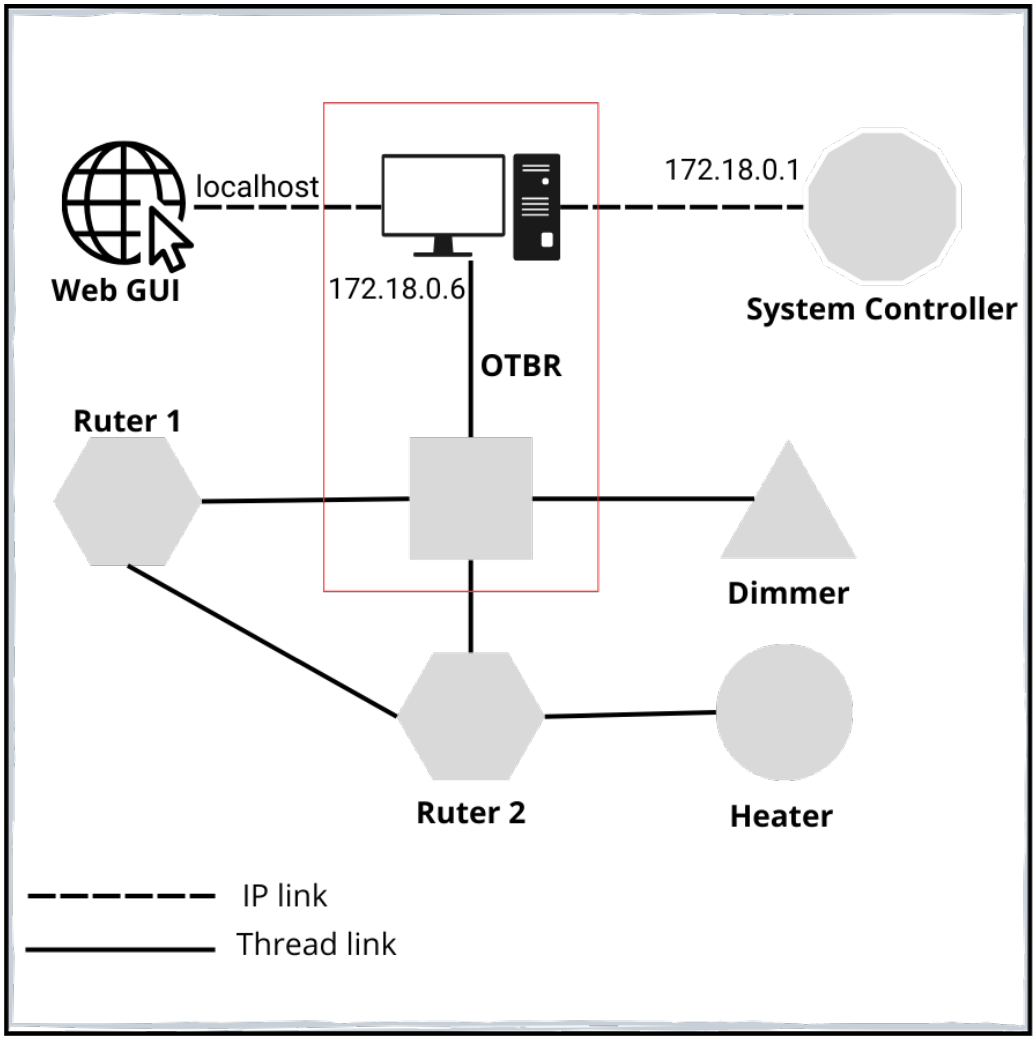
Ukończywszy proces Commissioning, dokonano analizy powstałej topologi, dzięki której ustalono, że w sieci funkcjonują 2 węzły typu Dziecko, które pełnią role HEATER i DIMMER, oraz 3 Rutery, z których jeden jest Liderem i Ruterem Brzegowym OTBR. Na potrzeby rozróżnienia 2 pozostałych Ruterów w dalszej części rozdziału, nadano im nazwę *Router 1* oraz *Router 2*. Następnie zweryfikowano połączenie między każdym węzłem sieci, rozsyłając pakiet ICMP (ang. *Internet Control Message Protocol*) na adres multicast FF03::1, który jest subskrybowany przez każde urządzenie w danej sieci Thread. W Tabeli 5.2 zestawiono MLEID (ang. *Mesh-Local Endpoint Identifiers*), czyli unikalne adresy IPv6 węzłów w utworzonej sieci Thread.

Tabela 5.2. MLEID węzłów w sieci.

Nazwa Węzła	Adres IPv6
OTBR	fd3f:cd2f:14fe:87f9:6b09:304e:878:6fb1
HEATER	fd3f:cd2f:14fe:87f9:434d:bde5:7686:b429
DIMMER	fd3f:cd2f:14fe:87f9:7998:c2a3:d56d:e3b0
Router 1	fd3f:cd2f:14fe:87f9:13a4:eced:d2c3:3922
Router 2	fd3f:cd2f:14fe:87f9:ce9e:4840:7dba:7356

Upewniwszy się o poprawności funkcjonowania sieci, oraz że każde z urządzeń otrzymuje odpowiedź od pozostałych węzłów, uruchomiono aplikację System Controllera oraz Web GUI wykorzystując przygotowany wcześniej skrypt *my-run-coap-server.sh* znajdujący się w repozytorium [12].

Topologię systemu po uruchomieniu wszystkich jego komponentów przedstawiono na Rysunku 5.3.



Rys. 5.3. Topologia uruchomionego systemu.

Ostatecznie włączono aplikację HEATER oraz DIMMER poprzez wciśnięcie przycisku Button 3 na obydwu urządzeniach. W efekcie zostało nawiązane połączenie klient-serwer między HEATER i DIMMER a System Controllerem.

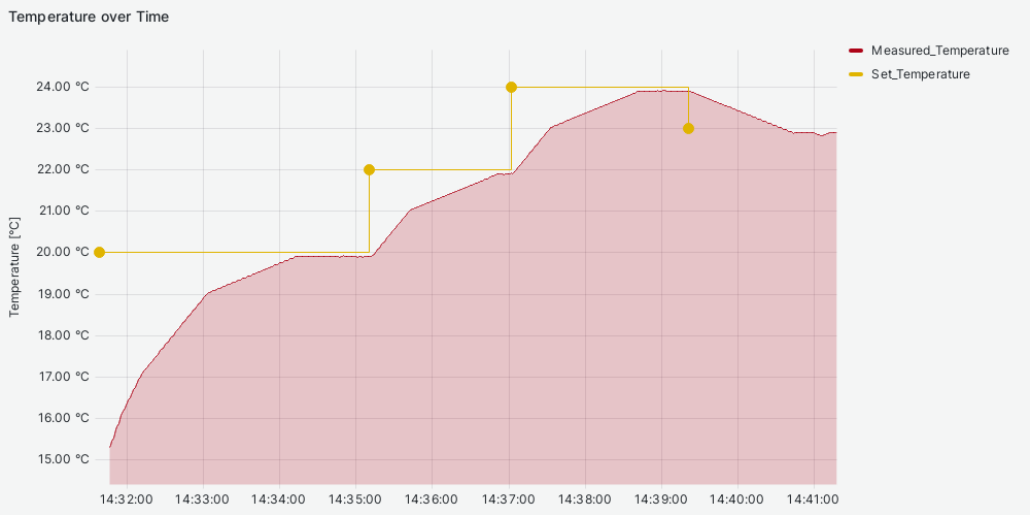
Na Rysunku 5.4 zaprezentowano zrzut ekranu z konsoli, w której uruchomiono program System Controllera.

```
Responding with heater_regulation.GET.Rsp

Received dimmer.PUT.Req

New current_illuminance value on the server side: 50.0

Responding with dimmer.PUT.Rsp

Received dimmer_regulation.GET.Req

Responding with dimmer_regulation.GET.Rsp

Received temperature.PUT.Req

New current_temp value on the server side: 17.7

Responding with temperature.PUT.Rsp

Received heater_regulation.GET.Req

Responding with heater_regulation.GET.Rsp
```

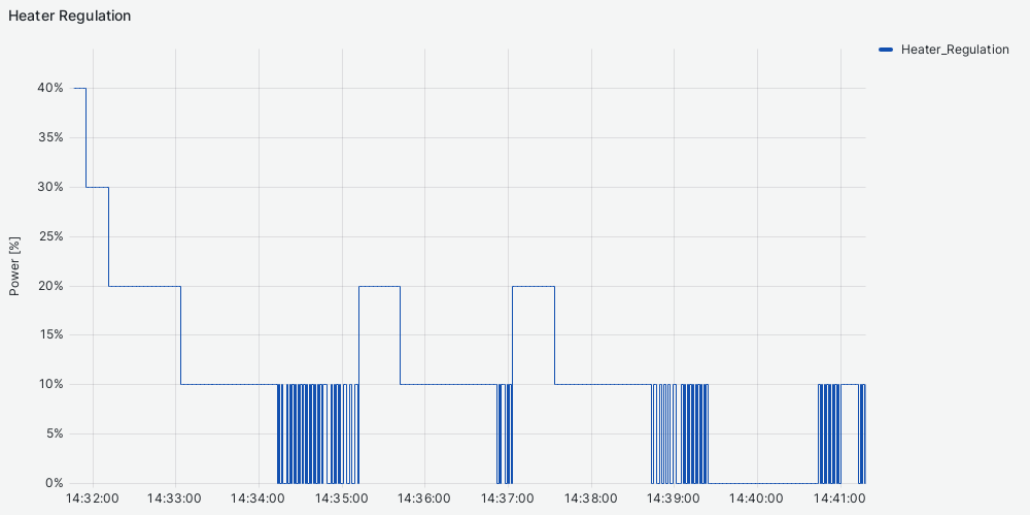
Rys. 5.4. Zrzut ekrany przedstawiający działanie System Controllera.

5.2. Weryfikacja zasady działania systemu

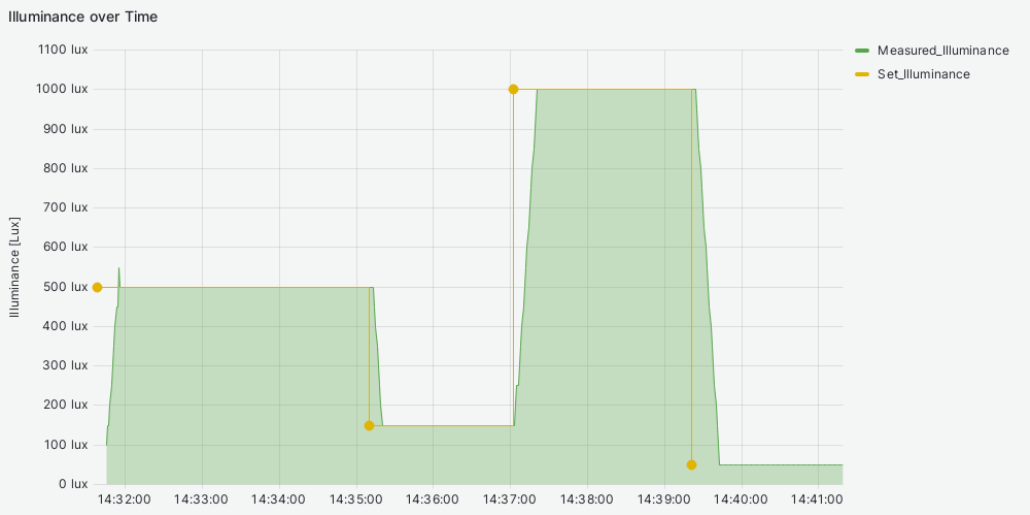
W celu weryfikacji poprawności zachowania stworzonego systemu wykorzystano narzędzie Grafana [19]. Platforma Grafana pozwala na tworzenie interaktywnych paneli do monitorowania danych z wybranego przez użytkownika źródła, takiego jak baza danych. Aby zwizualizować dane pochodzące z logów systemowych, jako zasób wybrano zaimplementowaną Bazę Danych, której architektura przedstawiono na Rysunku 4.2. Następnie z wykorzystaniem kwerend SQL utworzono panel Grafana zawierający 4 przebiegi czasowe, które przedstawiono na Rysunkach 5.5, 5.6, 5.7, 5.8.



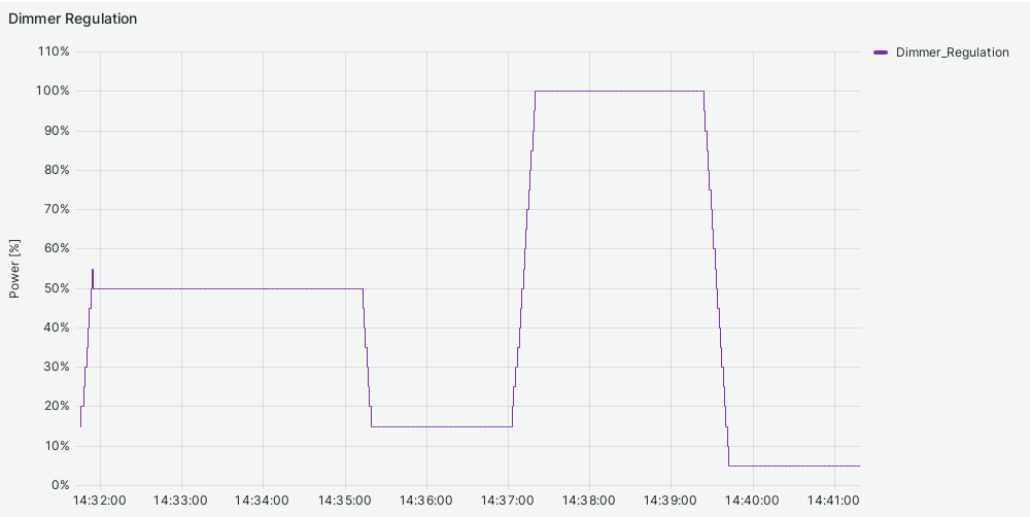
Rys. 5.5. Przebiegi czasowe zmierzonej oraz ustalonej temperatury.



Rys. 5.6. Przebieg czasowy regulacji mocy układu HEATER.



Rys. 5.7. Przebiegi czasowe zmierzonego oraz ustalonego natężenia oświetlenia.



Rys. 5.8. Przebieg czasowy regulacji mocy układu DIMMER.

Z przebiegów znajdujących się na Rysunku 5.5 oraz 5.7 można zauważyć, że odpowiednio aktualna temperatura oraz natężenie oświetlenia zmierzają do ustalonej przez użytkownika wartości. Zobrazowane na Rysunkach 5.6 oraz 5.8 zmiany parametru regulacji pokrywają się ze zdefiniowanymi algorytmami w System Controllerze. Zatem, obserwując wszystkie 4 wymienione wykresy, można stwierdzić o poprawności działania systemu, względem poczynionych założeń. Na Rysunku 5.6 można zauważyć skokowe zmiany obliczanej wartości parametru regulacji, które są konsekwencją zauważonego w Sekcji 4.2.3 problemu, związanego z brakiem histerezy w zaimplementowanym algorytmie System Controllera.

6. Analiza śladu pamięci

W celu umożliwienia estymacji wymaganych zasobów sprzętowych w systemach opartych o protokół Thread, a w szczególności korzystających z implementacji OpenThread na platformie nRF52, dokonano badania, w którym analizowano stopień wykorzystania pamięci RAM oraz ROM (ang. *Read-Only Memory*) po uruchomieniu stosu Thread oraz w zależności od wybranych funkcjonalności.

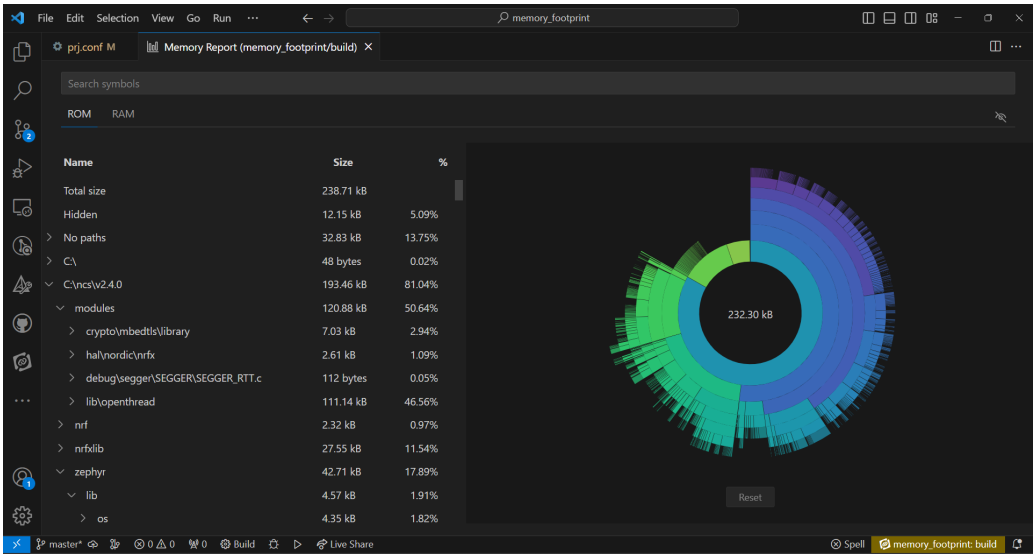
6.1. Doświadczenie

6.1.1. Założenia

Badanie przeprowadzono dla aplikacji dedykowanych platformie nRF52833 DK, stworzonych przy pomocy nRF Connect IDE. Do pomiarów wykorzystano narzędzia *Memory Raport*, które jest częścią wspomnianego środowiska programistycznego.

W ramach analizy dokonywano modyfikacji stworzonego na potrzeby doświadczenia projektu, poprzez włączanie kolejnych funkcjonalności powiązanych ze stosem Thread. W kolejnym kroku porównywano zużycie RAM oraz ROM, dodatkowo wyszczególniając obszary pamięci, które zajmują najwięcej miejsca.

Na Rysunku 6.1 przedstawiono zrzut ekranu prezentujący narzędzie Memory Raport.



Rys. 6.1. Zrzut ekranu z narzędziem Memory Raport.

6.1.2. Pomiar

6.1.2.1. Projekt bez dodatkowych konfiguracji

W celu ustalenia punktu odniesienia zbudowano projekt bez uwzględniania jakichkolwiek dodatkowych opcji konfiguracji. Uzyskane zużycie pamięci zestawiono na pozycji *1* w Tabeli 6.1.

Zauważono, że większość wykorzystanego ROM związana jest z uruchomieniem systemu operacyjnego Zephyr i jest skojarzona z uwzględnieniem sterowników sprzętowych oraz warstwy HAL (ang. *Hardware Abstraction Layer*). W przypadku pamięci RAM, większość zasobów została poświęcona na stos przerwań oraz na inicjalizację jądra systemu operacyjnego.

6.1.3. Projekt z opcją networking

Do inicjalizacji stosu Thread, niezbędnym jest skonfigurowanie warstwy sieciowej IP oraz funkcjonalności z nią związanych. W tym celu wzbogacono projekt o parametr konfiguracyjny *CONFIG_NETWORKING*, który uwzględnia w aplikacji warstwę łącza danych oraz warstwę sieciową z protokołem IP. Uzyskane zużycie pamięci zestawiono na pozycji *2* w Tabeli 6.1.

6.1.4. Projekt z włączonym stosem Thread v1.3

Po włączeniu warstwy sieciowej projekt jest przygotowany na wystartowanie protokołu Thread w implementacji OpenThread. Dodano do pliku konfiguracyjnego opcję *CONFIG_NET_L2_OPENTHREAD*, która konfiguruje LR-WPAN oraz uwzględnia funkcjonalności niezbędne do uruchomienia stosu Thread. Uzyskane zużycie pamięci zestawiono na pozycji *3* w Tabeli 6.1.

Uwzględnienie stosu Thread spowodowało ponad czterokrotny wzrost zużycia ROM oraz niemalże czterokrotnie większe wykorzystanie RAM.

Najwięcej zasobów pamięci nieulotnej jest pochłanianych przez bibliotekę OpenThread, uwzględnioną przez nRF Connect SDK. Rozmiar sekcji pamięci przeznaczonej dla biblioteki OpenThread wynosi 94,01 kB. Stos Thread wymaga odbioru ramek IEEE 802.15.4. Obszar dedykowany dla sterowników do konfiguracji LR-WPAN zajął 27,31 kB.

Wysokie zużycie RAM jest spowodowane koniecznością alokacji sekcji pamięci dla kontekstu sieci Thread, zastosowań kryptograficznych Mbed TLS oraz obsługi radia LR-WPAN. Sekcja przeznaczona na kontekst *ot::gInstanceRaw* zajmuje 27,44 kB, natomiast obszar wykorzystywany przez Mbed TLS zużywa 12,92 kB.

6.1.5. Projekt z włączonym stosem Thread w wersji 1.1 oraz 1.2

W celu porównania zużycia zasobów sprzętowych między kolejnymi wersjami Thread dokonano pomiaru wykorzystania RAM oraz ROM również dla wersji 1.1 oraz 1.2., dodając do pliku konfiguracyjnego parametr *CONFIG_OPENTHREAD_THREAD_VERSION_1_1* lub
CONFIG_OPENTHREAD_THREAD_VERSION_1_2. Uzyskane zużycie pamięci zestawiono na pozycji *4* oraz *5* w Tabeli 6.1.

Niewielka różnica w wielkości wykorzystanych zasobów RAM oraz ROM między wersją 1.1 a 1.2, oraz 1.3 wynika z konieczności uwzględnienia dodatkowych mechanizmów i funkcjonalności w sieci Thread [20], takich jak *Coordinated Sampled Listening*, *Link Metrics Probing*, *Multicast across Thread networks*, *Thread Domain unicast addressing*, *Enhanced Frame Pending, Enhanced Keep Alive*.

Ostatecznie, dla kolejnych rozważań, przywrócono konfigurację stosu Thread do wersji 1.3.

6.1.6. Projekt z włączonym stosem Thread dla różnych typów urządzeń

Domyślnym typem urządzenia dla skonfigurowanej aplikacji z włączonym stosem Thread jest FTD. W celu zbadania zużycia zasobów pamięci dla projektu, w którym urządzenie działa jako MTD, nadpisano konfigurację, dodając do pliku konfiguracyjnego parametr *CONFIG_OPENTHREAD_MTD*. Co więcej, zmieniono konfigurację projektu, dodając opcję *CONFIG_OPENTHREAD_MTD_SED*, aby zbadać zmianę użytych zasobów dla urządzenia SED. Uzyskane zużycie pamięci zestawiono na pozycji *6* oraz *7* w Tabeli 6.1.

Można zauważyć, że przełączenie konfiguracji na MTD obniża zużycie zarówno ROM jak i RAM. Urządzenia typu MTD posiadają zredukowany zestaw funkcjonalności w sieci Thread, jak np. nie są w stanie przekazywać pakietów do innych węzłów. Potwierdza to zmniejszona, w stosunku do przypadku 5, sekcja pamięci nieulotnej, dedykowana dla biblioteki OpenThread, której rozmiar wynosi 64,07 kB, jak również redukcja rozmiaru kontekstu sieci *ot::gInstanceRaw* do wielkości 18,48 kB.

Na skutek dodania funkcjonalności SED, nie zmierzono zmian w zużyciu zasobów pamięci.

Na koniec przywrócono w projekcie rolę FTD.

6.1.7. Projekt z włączonym stosem Thread oraz skonfigurowanym mechanizmem Commissioning

Projekt ze skonfigurowanym stosem Thread w wersji 1.3 wzbogacono o funkcjonalności związane z mechanizmem Commissioning. Do pliku konfiguracyjnego dodano opcję
CONFIG_OPENTHREAD_COMMISSIONER, aby ustalić rolę Commissioner urządzenia, a następnie *CONFIG_OPENTHREAD_JOINER* w celu konfiguracji roli Joiner. Uzyskane wartości zużytej pamięci zestawiono na pozycji *8* oraz *9* w Tabeli 6.1.

Po dokonaniu pomiaru wyłączono mechanizm Commissioning.

6.1.8. Projekt z włączonym stosem Thread oraz ze skonfigurowaną konsolą OpenThread CLI

Aby umożliwić konfigurację urządzeń w sieci Thread poprzez użycie OpenThread CLI, niezbędne jest skonfigurowanie konsoli Shell. Utworzono nowy projekt, w którym skonfigurowano tylko konsolę. Zmierzono zużycie zasobów i zestawiono na pozycji *10* w Tabeli 6.1. Kolejno powrócono do projektu ze stosem Thread v1.3 dla urządzenia FTD, a następnie włączono funkcjonalności konsoli Shell oraz dodatkowo skonfigurowano OpenThread CLI. Wykorzystany RAM oraz ROM po uwzględnieniu w projekcie OpenThread CLI zestawiono na pozycji *11* w Tabeli 6.1.

Tabela 6.1. Zużycie zasobów pamięci dla kolejno omwaianych przypadków.

l.p.	ROM [kB]	RAM [kB]	Opis
1	19,11	5,78	Brak dodatkowych konfiguracji
2	51,39	18,81	Konfiguracja L2 oraz L3 IP
3	217,18	72,76	Stos Thread v1.3, FTD
4	217,18	72,76	Stos Thread v1.2, FTD
5	211,20	71,46	Stos Thread v1.1, FTD
6	177,44	63,80	Stosu Thread v1.3, MTD
7	177,44	63,80	Stos Thread v1.3, MTD, SED
8	251,26	76,84	Stos Thread v1.3, FTD, Commissioner
9	245,96	76,52	Stos Thread v1.3, FTD, Joiner
10	42,99	11,44	Konfiguracja konsoli Shell
11	304,32	87,43	Stos Thread v1.3, FTD Openthread CLI

W Tabeli 6.2 przedstawiono oszacowane wartości zużytych RAM oraz ROM dla analizowanych funkcjonalności, które zostały obliczone na podstawie różnic między wartościami zasobów umieszczonych w Tabeli 6.1.

Tabela 6.2. Zużycie zasobów pamięci przez wybrane komponenty.

Funkcjonalność	ROM [kB]	RAM [kB]	Wymagana funkcjonalność
Inicjalizacja Zephyr	19,11	5,78	brak
Networking	32,28	13,03	Inicjalizacja Zephyr
Konsola Shell	23,88	5,66	Inicjalizacja Zephyr
Stos Thread v1.3 dla FTD	165,79	53,95	Networking
Stos Thread v1.3 dla MTD	126,05	44,99	Networking
Rola Commissioner dla FTD	40,06	4,08	Stos Thread v1.3 dla FTD
Rola Joiner dla FTD	28,16	3,76	Stos Thread v1.3 dla FTD
OpenThread CLI dla FTD	44,15	12,19	Stos Thread v1.3 dla FTD, Konsola Shell

6.2. Wnioski

Przeprowadziwszy analizę zużycia zasobów pamięci dla funkcjonalności protokołu Thread, na platformie nRF52833 w implementacji Openthread, wysunięto następujące wnioski.

Uruchomienie OpenThread wymaga dołączenia biblioteki, która w stosunku do innych wykorzystanych sekcji ROM, wymaga dużej liczby dostępnej pamięci nieulotnej. Nie jest możliwym na podstawie przeprowadzonej obserwacji, aby stwierdzić, czy uwzględnienie całej biblioteki załączonej do pamięci nieulotnej, jest

niezbędne do poprawnego działania stosu Thread. Jeżeli w użytej bibliotece są zawarte fakultatywne funkcjonalności, istniałaby możliwość na optymalizację zużycia zasobów, poprzez selektywny dobór tylko tych funkcjonalności, który są konieczne do zapewnienia zgodnego ze standardem [3] działania sieci Thread.

Największe zużycie RAM spowodowane jest potrzebą zaalokowania w pamięci ulotnej obszaru dla kontekstu sieci Thread oraz zmiennych wykorzystywanych w zastosowaniach kryptograficznych. Domyślną biblioteką algorytmów szyfrowania, która została uwzględniona przy uruchomieniu OpenThread, jest Mbed TLS. W celu ewentualnej optymalizacji zużytych zasobów pamięci, sugerowane jest porównanie wykorzystanej implementacji z innymi, otwartymi, dostępnymi bibliotekami TLS (ang. *Transport Layer Security*), takimi jak np. *OpenSSL*, *libsodium*, *Crypto++* [21].

Rola MTD wymaga mniejszych zasobów niż FTD. Urządzenia końcowe pracujące jako MED, pełniące rolę czujników i regulatorów, są w stanie posiadać mniejsze zasoby pamięci, niezbędne do poprawnego działania. Z kolei do uruchomienia urządzeń pracujących jako Rutery, konieczne jest uwzględnienie większego zaplecza pamięci. Ponadto włączenie roli Commissioner zużywa więcej RAM oraz ROM, aniżeli uruchomienie roli Joiner. Urządzeniem Commissioner może być tylko Ruter, co wprowadza dodatkowe wymagania odnośnie dostępnej pamięci.

W przypadku potrzeby wykorzystywania konsoli OpenThread CLI, zalecane jest, aby wykorzystać urządzenie o pamięci RAM oraz ROM większej o liczbę przedstawioną w Tabeli 6.2. Włączenie konsoli OpenThread nie jest niezbędne, natomiast może być pomocna w przypadku chęci dynamicznej konfiguracji sieci Thread w trakcie działania aplikacji, szczególnie na takich urządzeniach jak Rutery czy Rutery Brzegowe.

Podsumowanie

Podczas realizacji pracy, wykonano kolejne kroki:

1. Zapoznano się z charakterystyką stosu Thread i podstawowymi mechanizmami sieci Thread.
2. Zaprojektowano prototypowy system automatyki domowej, dokonując rozdziału funkcjonalności na poszczególne komponenty.
3. Zaznajomiono się z platformą sprzętową Nordic Semiconductor nRF52833 oraz narzędziami dostarczonymi przez producenta, jak również z otwartą, wolną, implementacją stosu Thread Openthread.
4. Stworzono w pełni funkcjonalną sieć Thread złożoną z 5 urządzeń platformy nRF52833. Uwzględniono mechanizm bezpieczeństwa Commissioning oraz zagwarantowano połączenie sieci z domeny Thread z zewnętrzną siecią Ethernet poprzez uruchomienie aplikacji Rutera Brzegowego.
5. Zaimplementowano aplikację zaproponowanego systemu, według poczynionych założeń. Zapewniono komunikację klient-serwer z wykorzystaniem protokołu CoAP. Zrealizowano pozostałe komponenty takie jak Baza Danych oraz Web GUI, po wcześniejszym zapoznaniu się z wykorzystywanymi technologiami.
6. Uruchomiono sieć Thread i sprawdzono połączenia między węzłami.
7. Wystartowano aplikację systemu automatyki domowej oraz zweryfikowano poprawność działania poprzez analizę wykresów wygenerowanych za pośrednictwem narzędzia Grafana.
8. Zbadano wykorzystanie zasobów pamięci ulotnej oraz nieulotnej przez protokół Thread, oraz jego funkcjonalności dla platformy nRF52833 z implementacją OpenThread.

W związku z charakterem pracy, której autor samodzielnie dokonywał implementacji oraz ograniczeniem czasowym przeznaczonym na realizację, zaprojektowany system ma charakter prototypu. W celu polepszenia funkcjonalności systemu oraz przygotowania go do wdrożenia, zaprojektowany system mógłby zostać wzbogacony o dodatkowe funkcjonalności, które wymieniono poniżej:

– Wprowadzenie dodatkowych mechanizmów bezpieczeństwa poprzez zastosowanie warstwy DTLS dla gniazd UDP, wykorzystywanych do transmisji zapytań i odpowiedzi CoAP.

– Zmiana mechanizmu On-mesh Commissioning na External Commissioning, wykorzystując implementację OpenThread. Stworzenie aplikacji mobilnej, który umożliwiałaby uwierzytelnienie węzłów Joiner poprzez zbliżenie anten NFC (ang. *Near-Field Communication*).

– Zamiana prototypowych algorytmów regulacji po stronie System Controllera na algorytmy wykorzystywane w automatyce przemysłowej, takie jak np. PID.

– Wprowadzenie do urządzeń końcowych HEATER oraz DIMMER fizycznych czujników oraz układów regulacji, w celu weryfikacji działania systemu w rzeczywistym środowisku domowym oraz dostosowania algorytmów System Controllera do istniejących warunków budynkowych.

– Przeniesienie aplikacji OTBR z Laptopa Dell na platformę Raspberry Pi, w celu ograniczenia zużycia mocy oraz zapewnienia możliwość ulokowania urządzenia w dowolnym miejscu w docelowym budynku.

– Migracja serwisu Web GUI wraz z Bazą Danych oraz System Controllera do chmury, aby umożliwić użytkownikowi zdalną konfigurację parametrów systemu. Jako technologię chmurową zastosowana mogłaby zostać platforma *Microsoft Azure*.

– Rozwinięcie Web GUI poprzez dodanie panelu, w którym umieszczone zostałyby dynamiczne generowane przebiegi czasowe z narzędzia Grafana.

Opracowanie sieci Thread oraz jej wdrożenie pozwoliły na przybliżenie charakterystyki stosu protokołów opracowanego przez Thread Group. Wykorzystanie mechanizmu Commissioning potwierdziło tezę o wysokim, w stosunku do innych protokołów sieci mesh (takich jak Bluetooth Mesh oraz Zigbee), poziomie bezpieczeństwa. Cecha samoleczenia oraz dynamiczne zarządzanie rolami i topologią sieci przez Lidera Thread, mityguje skutki awarii urządzeń w sieci. Integracja sieci domeny Thread z inną zewnętrzną siecią IP odbywa się poprzez włączenie do sieci urządzenia o zestandaryzowanej roli Rutera Brzegowego. Czyni to wdrażanie systemów IoT funkcjonujących z różnymi technologiami IP prostszym i skuteczniejszym. Takie zachowania stosu Thread mogą nieść szereg korzyści w zastosowaniach automatyki domowej i budynkowej. Dotyczy to szczególnie przypadku systemów, w których funkcjonuje duża liczba czujników oraz regulatorów, potrzebujących wymieniać pakiety z innymi, zewnętrznymi sieciami IP. Co więcej, możliwość integracji sieci Thread z Internetem, mogłaby ułatwić przeniesienie serwisów automatyki domowej lub budynkowej do chmury.

Analiza zużycia zasobów pamięci, w przypadku uruchomienia aplikacji działającej na stosie Thread, zwróciła uwagę na istotę estymacji zasobów platformy sprzętowej, przed przystąpieniem do tworzenia aplikacji dla systemów wbudowanych. Wymagania dotyczące wielkości RAM oraz ROM, podczas uruchomienia Thread oraz funkcjonalności z nim związanych, rzuciły cień na problem wdrożenia tego typu sieci w urządzeniach o małej liczbie pamięci ulotnej oraz nieulotnej.

Bibliografia

[1] The Verge. *Thread is Matter's secret sauce for a better smart home*. *https://www.theverge.com/23165855/thread-smart-home-protocol-matter-apple-google-interview*. Online; dostęp 15:34 4.12.2023.

[2] Silicon Labs. *Benchmarking Bluetooth Mesh, Thread, and Zigbee Network Performance*. *https://www.silabs.com/wireless/multiprotocol/mesh-performance*. Online; dostęp 16:46 4.12.2023.

[3] Thread Group. *Thread 1.3.0 Public Specification*. Wer. 1.3.0.

[4] Openthread. *Node Roles and Types*. *https://openthread.io/guides/thread-primer/node-roles-and-types*. Online; dostęp 17:20 3.12.2023.

[5] Nordic Semiconductor. *OpenThread commissioning*. *https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/thread/overview/commissioning.html*. Online; dostęp 13:10 5.12.2023.

[6] Nordic Semiconductor. *nRF52833 Product Specification*. *https://infocenter.nordicsemi.com/topic/ps_nrf52833/keyfeatures_html5.html?cp=5_1_0*. Online; dostęp 13:04 8.12.2023.

[7] Nordic Semiconductor. *nRF52833 DK Block diagram*. *https://infocenter.nordicsemi.com/topic/ug_nrf52833_dk/UG/dk/hw_block_diagram.html?cp=5_1_4_7_1*. Online; dostęp 12:11 8.12.2023.

[8] Nordic Semiconductor. *nRF Connect SDK*. *https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/index.html*. Online; dostęp 13:49 8.12.2023.

[9] Nordic Semiconductor. *nRF Connect for VS Code*. *https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-vs-code*. Online; dostęp 14:45 8.12.2023.

[10] Google. *OpenThread Nordic Semiconductor platforms*.
https://openthread.io/vendors/nordic-semiconductor.
Online; dostęp 16:51 8.12.2023.

[11] Google. *OpenThread Border Router*.
https://openthread.io/guides/border-router.
Online; dostęp 17:01 8.12.2023.

[12] Paweł Tymoczko. *Projekt systemu automatyki domowej.*
https://github.com/Pawlicho/thread-coap.
Online; dostęp: 12:51 10.12.2023.

[13] Google. *Run OTBR Docker*.
https://openthread.io/guides/border-router/docker/run.
Online; dostęp: 11:15 10.12.2023.

[14] Google. *OpenThread on Nordic NRF528xx Example*.
https://github.com/openthread/ot-nrf528xx. Online; dostęp: 11:20 10.12.2023.

[15] The SQLite Consortium. *SQLite overview.*
https://www.sqlite.org/index.html. Online; dostęp: 14:15 10.12.2023.

[16] Christian Amsüss i the aiocoap contributors. *aiocoap – The Python CoAP library.*
https://aiocoap.readthedocs.io/en/latest.
Online; dostęp: 14:15 10.12.2023.

[17] Flask. *Flask's Documentation*.
https://flask.palletsprojects.com/en/3.0.x/.
Online; dostęp: 17:20 8.12.2023.

[18] Google. *OpenThread CLI Overview*.
https://openthread.io/reference/cli.
Online; dostęp: 12:18 10.12.2023.

[19] Grafana Labs. *Grafana*.
https://grafana.com/grafana/.
Online; dostęp: 9:38 12.12.2023.

[20] Nordic Semiconductor. *Thread Specification options*.
https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/thread/configuring.html#id11.
Online; dostęp: 11:22 15.12.2023.

[21] LibHunt Network Awesome C++. *Mbed TLS alternatives*.
https://cpp.libhunt.com/mbedtls-alternatives.
Online; dostęp: 10:07 16.12.2023.