



Politechnika
Wrocławska

Projekt zespołowy

Kierunek	<i>Informatyka</i>	Termin	<i>Czwartek 14:15</i>
Temat	<i>Projekt elastycznej aplikacji do zarządzania urządzeniami IoT w oparciu o bibliotekę QT</i>	Zgłaszący	<i>InterElcom</i>
Skład grupy	<i>Adam Krizar 241276 Katarzyna Czajkowska 242079 Mateusz Gurski 242089 Szymon Cichy 235093 Arkadiusz Cichy 236011</i>	Nr grupy	-
Prowadzący	<i>Dr inż. Jan Nikodem</i>	data	<i>21 maj 2020</i>

Spis treści

Spis treści.....	2
1. Plan zadań.....	5
2. Opis zadania	5
3. Wymagania.....	6
4. Założenia	6
5. Rysunek ogólny	6
6. Opis systemu	8
7. Środowisko	10
7.1. Instalacyjne.....	10
7.2. Programistyczne	10
8. Wybrane urządzenia/czujniki	10
8.1. Wstęp.....	10
8.2. Pula kontrolerów	10
8.3. Pula czujników	13
8.4. Wybór kontrolera.....	16
8.5. Wybór czujnika	16
8.6. Schemat elektryczny.....	17
8.7. Sposób programowania	17
8.8. Oficjalna dokumentacja.....	18
9. Wybrane warstwy OSI.....	18
9.1. Model OSI	18
9.2. Warstwa aplikacji	19
9.3. Warstwa transportowa	20
9.4. Warstwa sieci.....	21
9.5. Routing	22
10. Transmisja WiFi oraz TCP/IP	22
10.1. Transmisja WiFi	22
10.2. TCP/IP	23
10.3. TCP a UDP	23
10.4. HTTP	24
10.5. MQTT	24
11. Podział na podsieci	25
11.1. Teoria podsieci	25
11.2. Adresacja	26
12. Implementacja obsługi protokołu HTTP w aplikacji mobilnej	26
12.1. Test obsługi protokołu HTTP w aplikacji mobilnej	27
12.2. Test obsługi protokołu HTTP w aplikacji mobilnej przy wykorzystaniu trybu administratora	28
13. Implementacja obsługi protokołu HTTP w aplikacji desktopowej	30

13.1.	Omówienie podstawowych funkcji modułu realizującego obsługę protokołu HTTP	31
13.2.	Test obsługi protokołu http w aplikacji desktopowej	32
14.	Implementacja obsługi protokołu MQTT w aplikacji desktopowej	34
14.1.	Obsługa protokołu MQTT	34
14.2.	Kluczowe funkcje biblioteki.....	34
14.3.	Pozostałe funkcje biblioteki.....	35
15.	Oprogramowanie techniczne WiFi (Aplikacja użytkownika na system Android)	35
15.1.	Przeznaczenie	35
15.2.	Uruchomienie	37
15.2.1.	Środowisko Programistyczne	37
15.2.2.	Uruchamianie aplikacji.....	38
15.3.	Schemat programu	41
15.4.	Opis oprogramowania z komentarzami i zrzutami ekranu	42
15.5.	Uwagi rozwojowe.....	47
16.	Oprogramowanie administratora – WiFi (Tryb administratora w aplikacji na system android).....	47
16.1.	Przeznaczenie	47
16.2.	Uruchomienie	49
16.3.	Schemat programu	50
16.4.	Opis oprogramowania z komentarzami i zrzutami ekranu	52
16.5.	Uwagi rozwojowe.....	56
17.	Oprogramowanie użytkownika Router WiFi (Tryb użytkownika w aplikacji na system Linux).....	56
17.1.	Przeznaczenie	56
17.2.	Uruchomienie	58
17.2.1.	Uruchomienie Linuxa w Oracle Virtual Box.....	58
17.2.2.	Uruchomienie systemu Linux na komputerze	63
17.2.3.	Uruchomienie aplikacji.....	64
17.3.	Schemat programu	67
17.4.	Opis oprogramowania wraz z komentarzami i zrzutami ekranów	69
17.4.1.	Podstawowe funkcje aplikacji desktopowej	69
17.4.2.	Pasek Menu	75
17.5.	Uwagi rozwojowe.....	78
18.	Oprogramowanie administratora – Router WiFi (Tryb administratora w aplikacji na system Linux)...	78
18.1.	Przeznaczenie	78
18.2.	Uruchomienie	80
18.3.	Schemat programu	80
18.4.	Opis oprogramowania i komentarze.....	81
18.5.	Uwagi rozwojowe.....	83
19.	Oprogramowanie czujników (urządzeń IoT).....	83
19.1.	Przeznaczenie	83

19.2.	Schemat połączeń	84
19.4.	Uruchomienie	86
19.4.	Schemat programu	88
19.5.	Opis oprogramowania z komentarzami	90
19.6.	Uwagi rozwojowe.....	93
20.	Wsparcie użytkownika	94
20.1.	Tryb programistyczny.....	94
20.2.	Użytkownik czujników.....	94
21.	Kosztorys.....	95
22.	Plan realizacji	95
23.	Propozycja rozwoju systemu.....	95
24.	Źródła	97

1. Plan zadań

- **Opis projektu, wymagania, założenia, rysunek ogólny, opis systemu, PREZENTACJA PowerPoint:**

Wykonanie: Adam Krizar

- **Oprogramowanie czujników:**

Wykonanie: Mateusz Gurski

Sprawdzenie: Szymon Cichy

- **Oprogramowanie techniczne WiFi:**

Wykonanie: Mateusz Gurski

Sprawdzenie: Katarzyna Czajkowska

- **Oprogramowanie użytkowe, router WiFi:**

Wykonanie: Mateusz Gurski

Sprawdzenie: Arkadiusz Cichy

- **Oprogramowanie użytkowe, aplikacja HTTP/MQTT:**

Wykonanie: Szymon Cichy

Sprawdzenie: Arkadiusz Cichy

- **Oprogramowanie administratora Android, router WiFi:**

Wykonanie: Arkadiusz Cichy

Sprawdzenie: Mateusz Gurski

- **Oprogramowanie administratora Internet:**

Wykonanie: Katarzyna Czajkowska

Sprawdzanie: Szymon Cichy

2. Opis zadania

Wykonanie: Adam Krizar

Naszym zadaniem jest stworzenie aplikacji, która umożliwia komunikację z urządzeniami IoT zezwalając na zmianę protokołu komunikacji (elastyczność).

Stan początkowy określa jedynie platformy, które mamy wspierać oraz technologie które mają być wykorzystane do komunikacji z urządzeniem IoT. Ze względu na bardzo mało precyzyjny opis wielu parametrów projektu jesteśmy zmuszeni samodzielnie doprecyzować wiele rzeczy takich jak na przykład wykorzystane protokoły sieciowe. Naszym zadaniem jest więc określenie następujących rzeczy:

- W jakiej wersji wykorzystać wymagane narzędzia.
- Określić w jakim środowisku oraz w jaki sposób urządzenia IoT będą komunikować się z naszą aplikacją.
- Zaprojektowanie dwóch aplikacji na dwa środowiska (mobilne oraz PC)
- Określenie wymagań sieci, jak powinna być skonfigurowana i jakie wykorzystywać urządzenia.
- Zdobycie informacji na temat wykorzystywanych protokołów, jak je obsłużyć oraz zaprogramować na różnych platformach.
- Określenie czujnika oraz rodzaju mikontrolera, który będzie służył do prezentacji możliwości naszej aplikacji.
- Przygotowanie oprogramowania dla testowanego urządzenia, które pozwoli mu współpracować z naszą aplikacją.

3. Wymagania

Wykonanie: Adam Krizar

Celem projektu jest utworzenie aplikacji działającej na kilku platformach w oparciu o bibliotekę QT i język C++. Jej elastyczność będzie polegała na możliwości zmiany protokołu komunikacji z urządzeniem IoT.

Wymagania, które powinna ona spełniać to:

- Użycie biblioteki QT oraz języka C++
- Stworzenie dwóch aplikacji działających minimum na dwóch platformach (np. Linux, Android).
- Stworzenie w aplikacji możliwości wyboru oraz sposobu dodawania nowych protokołów komunikacji z urządzeniem IoT.
- Obsługa w aplikacji minimum dwóch protokołów komunikacji z urządzeniem IoT (np. HTTP, MQTT).

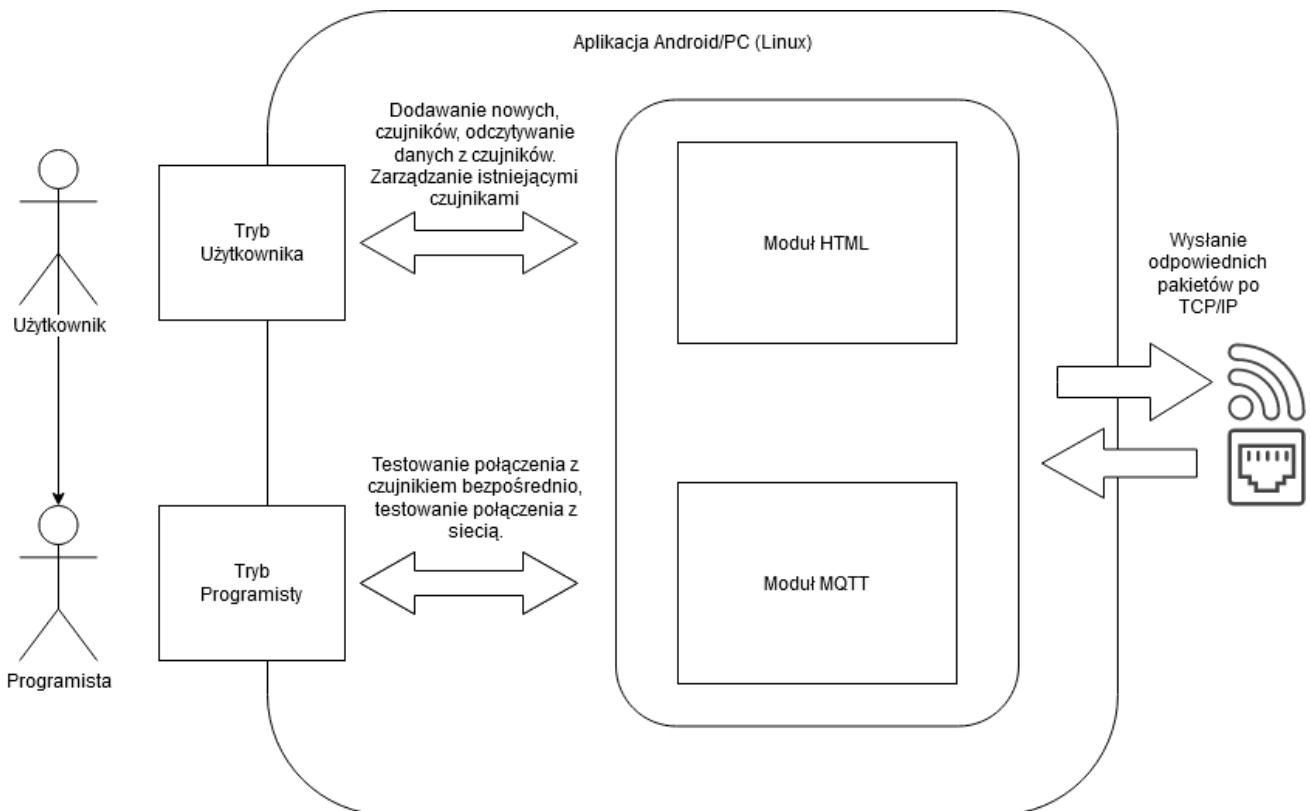
4. Założenia

Wykonanie: Adam Krizar

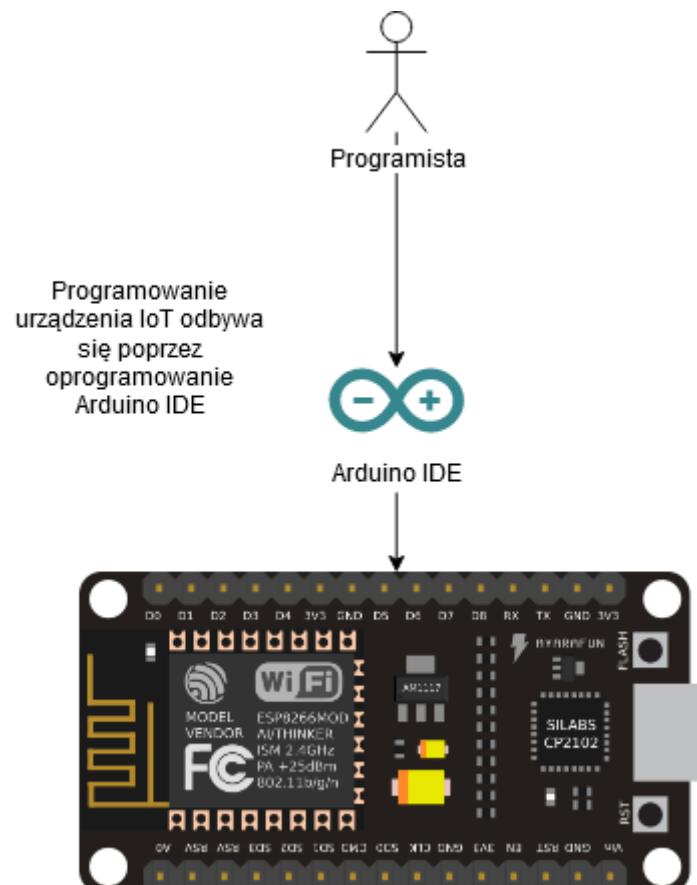
Bazując na zgłoszonych wymaganiach opracowaliśmy następujące cele naszego projektu:

- Wymaganie wykorzystania biblioteki Qt: Wykorzystanie Qt w wersji 5.14 wzwyż – Zapewnia wykorzystanie jak najdokładniejszych rozwiązań oraz gwarantuje dobre działanie na nowych systemach operacyjnych.
- Wymaganie dwóch aplikacji z różnymi trybami: użytkowy i programisty.
- Wymaganie obsługi dwóch platform: Wsparcie dla systemu Linux (ze względu na jego darmowość i łatwość instalacji na różnych urządzeniach) oraz dla systemu Android (obecnie najpopularniejsza platforma na urządzenia mobilne).
- Wsparcie dla systemu Android: Wykorzystanie pakietu Android Studio do stworzenia aplikacji na platformę mobilną firmy Google.
- Wymaganie implementacji minimum dwóch protokołów komunikacji z IoT: Implementacja protokołu HTTP oraz MQTT w naszej aplikacji oraz w testowym urządzeniu IoT. Te dwa protokoły zostały wyszczególnione jako przykładowe przez zgłaszającego oraz należą do najpopularniejszych rozwiązań na rynku co zapewni większą kompatybilność aplikacji.
- Wymaganie elastycznej aplikacji: Możliwość wyboru używanego protokołu komunikacji oraz przygotowanie możliwości dodania obsługi nowych protokołów
- Komunikacja z IoT: Aplikacja będzie realizować komunikacje poprzez sieć lokalną, która może odbywać się po kablu lub bezprzewodowo z wykorzystaniem protokołu TCP/IP.
- Możliwość obsługi wielu IoT: Projekt aplikacji przewiduje obsługę do 80 urządzeń. Ta liczba zależy od możliwości wybranego routera obsługującego połączenia.
- Przygotowanie dwóch urządzeń IoT (Wykorzystanie gotowych rozwiązań takich jak mikrokontrolery Arduino i im podobne) w celu prezentacji możliwości aplikacji.

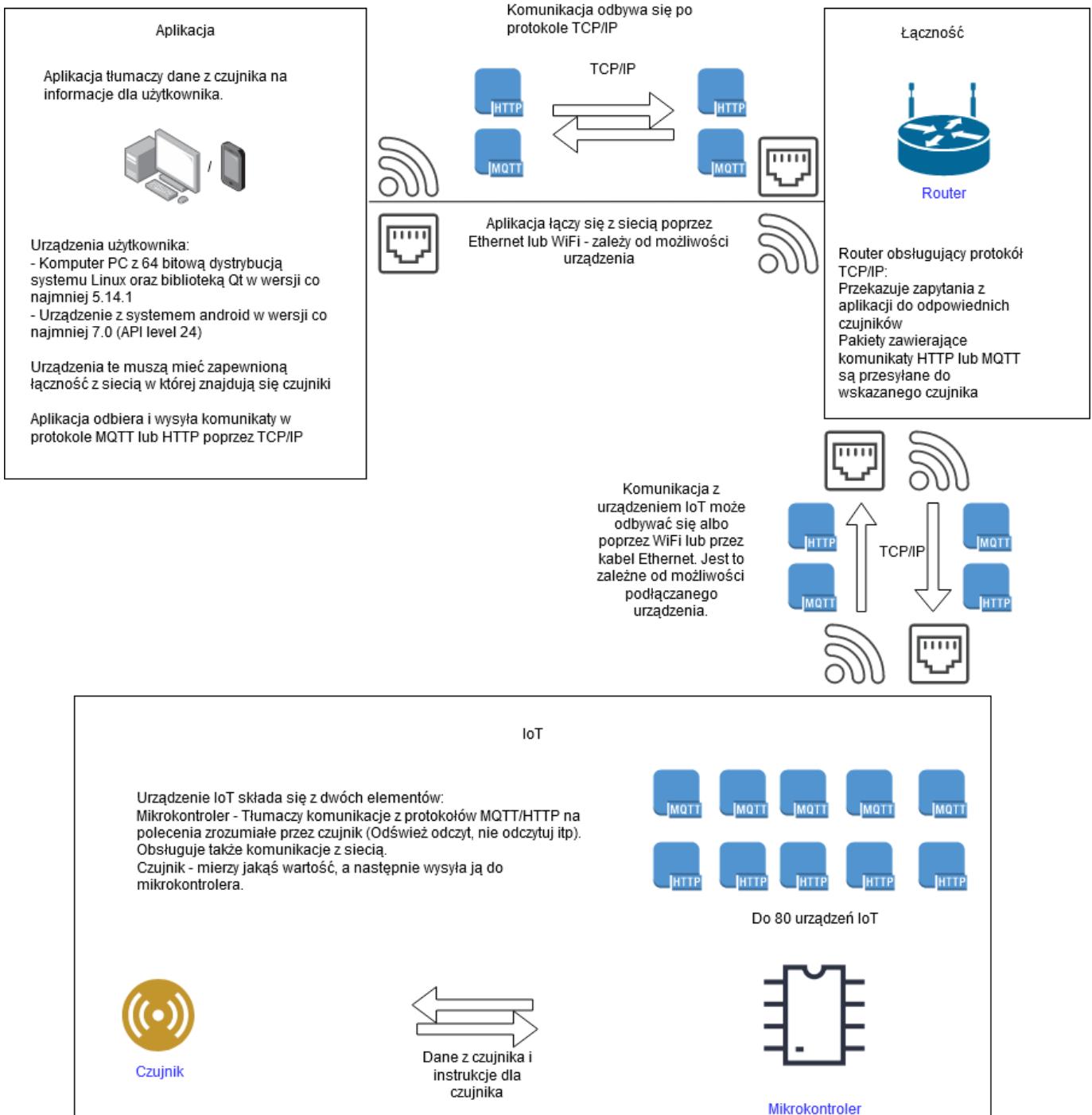
5. Rysunek ogólny



Rysunek 1. Ogólny schemat aplikacji.



Rysunek 2. Schemat programowanie urządzenia IoT.



Rysunek 3. Ogólny schemat

6. Opis systemu

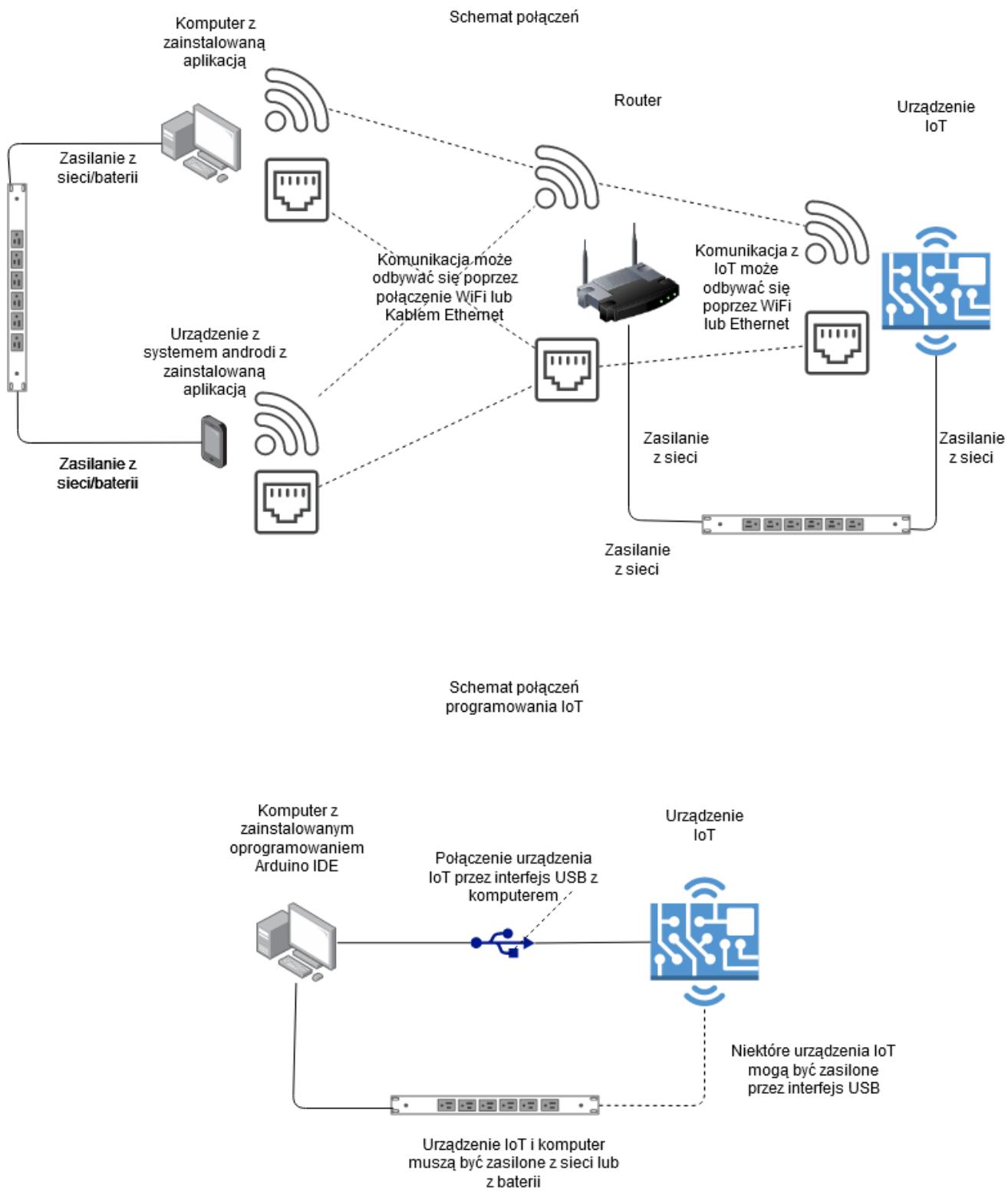
Aplikacja na system android komunikuje się z czujnikami w sieci lokalnej poprzez protokół IoT. Wszystkie urządzenia w systemie muszą być zasilone. Połączenie może wykorzystywać sieć WiFi lub kablowe (Ethernet).

Na przykład:

- Laptop zainstalowaną aplikacją może połączyć się z routерem zarówno poprzez WiFi lub Ethernet.
- Urządzenie z systemem Android można podłączyć poprzez Ethernet wykorzystując odpowiedni adapter. Przykładowo: <https://www.x-kom.pl/p/446050-przejsciwka-i-tec-adapter-usb-c-rj-45-metal-gigabit-ethereum.html>.

Punktem centralnym układu jest Router, przez który komunikują się wszystkie czujniki oraz urządzenia z zainstalowaną aplikacją.

Programowanie urządzenia odbywa się poprzez komputer PC z oprogramowaniem Arduino IDE. Urządzenia łączą się poprzez interfejs USB (W przypadku wykorzystywanej [platki](#)).



Rysunek 4. Schemat połączeń

7. Środowisko

Wykonanie: Szymon Cichy

Sprawdzenie: Adam Krizar

7.1. Instalacyjne

Łączność między komputerami na których zainstalowana zostanie aplikacja a urządzeniami IoT będzie odbywać się przez sieć lokalną poprzez łącze przewodowe bądź z użyciem transmisji bezprzewodowej WiFi.

Wymagania sprzętowe dla naszej aplikacji są trudne do precyzyjnego określenia na etapie projektowym. Zakładamy jednak, że każdy sprzęt, na którym może działać nowoczesny system operacyjny (np. Android 8+, dystrybucje Linux tj. Ubuntu, Manjaro) będzie wystarczający.

7.2. Programistyczne

Do budowy aplikacji wykorzystany zostanie język C++ i biblioteki Qt.

Framework Qt zostanie wykorzystany w najnowszej stabilnej wersji (na dzień 12.03.2020 jest to 5.14.1). Jest to zestaw narzędzi które pozwolą na stworzenie różnych interfejsów użytkownika na osobnych platformach, które te interfejsy będą spójne wizualnie oraz będą mogły przystosowywać się do różnic w konkretnych urządzeniach, jak np. dopasowanie elementów do rozmiarów ekranu.

Dla mobilnej wersji naszej aplikacji zostanie wykorzystany pakiet Android Studio jako najlepiej przystosowany do współpracy z systemem android. Wymusza to nas wykorzystanie języka Java ale gwarantuje stabilność gotowej aplikacji oraz prostotę ewentualnych przyszłych modyfikacji kodu.

Do tworzenia aplikacji desktopowej użyte zostaną narzędzia Qt Creator oraz QT Designer. Użycie ich usprawni utrzymanie aplikacji oraz wprowadzanie zmian w przyszłości. Wykorzystanie tych specjalnych środowisk poprawi jakość oraz obniży czas wykonania aplikacji, ponadto może skutkować niższymi kosztami obsługi w wypadku konieczności wprowadzenia zmian w interfejsie użytkownika.

Kończąc, użycie bibliotek Qt pozwoli na stworzenie kodu aplikacji który w spójny sposób obsługuje nie tylko interfejs użytkownika, lecz także obsługę protokołów komunikacji z urządzeniami.

Po stronie urządzeń IoT kod będzie napisany w języku C++ lub być może, w zależności od bieżących potrzeb, w innym języku jak np. skrypt Lua.

Wybór innych narzędzi programistycznych może nastąpić w trakcie wykonywania projektu i ich lista może zostać uzupełniona w późniejszej dacie.

W celu ułatwienia pracy w grupie wykorzystany zostanie system kontroli wersji. Repozytorium zostanie utworzone na platformie Github. Jest to sposób na centralizację zasobów w projekcie i ułatwi śledzenie zmian i postępu przez nie tylko programistów, lecz także zleceniodawców.

8. Wybrane urządzenia/czujniki

Wykonanie: Arkadiusz Cichy

Sprawdzenie: Szymon Cichy

8.1. Wstęp

Założenia projektowe sugerują wybór elektroniki o jak najmniejszym poborze mocy. Zadanie ułatwia fakt, że urządzenia nie muszą mieć dużej mocy obliczeniowej. Jedynym aspektem, który działa na naszą niekorzyść jest poziom skomplikowania programowania/łączenia wybranego sprzętu. Dysponując jedynie taką mocą przerobową, nasze wybory powinny uwzględniać czas nauki obsługi danego sprzętu dodatkowo do czasu zaprogramowania go lub czasu niezbędnego na zbudowanie działającego układu.

8.2. Pula kontrolerów

ESP8266

- **Komunikacja WiFi:**

- standard 802.11 b/g/n 2,4 GHz,
- prędkość transmisji do 72,2 Mb/s,
- zabezpieczenia: WPA/WPA2,
- szyfrowanie: WEP/TKIP/AES,
- protokoły: IPv4, TCP/UDP/HTTP.

- **Zasilanie:**

- napięcie pracy: 2,5 – 3,6 V,
- napięcie zasilania: 4,8 – 12 V,
- średni pobór prądu: 80 mA,
- maksymalny pobór prądu: 800 mA.

- **Aktualizacja oprogramowania:**

- UART,
- OTA.

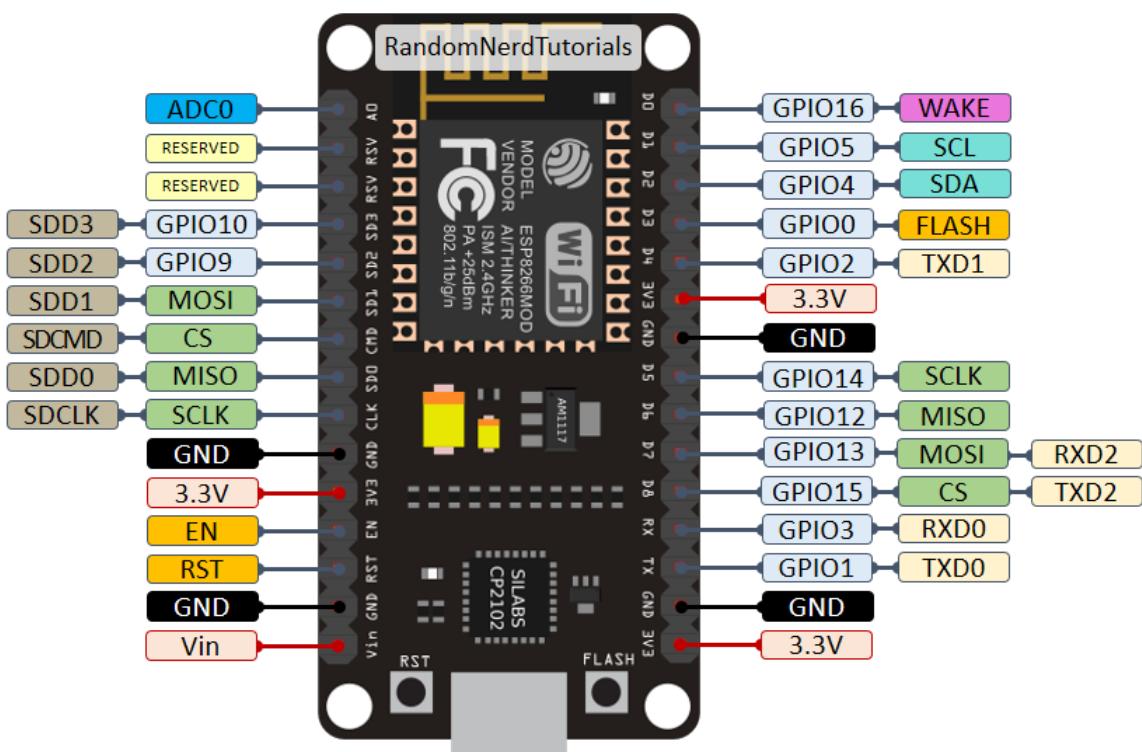
- **CPU:**

- Tensilica L106 32-bit 80 MHz,
- obudowa: QFN32-pin (5 mm × 5 mm),
- interfejsy: UART/SDIO/SPI/I2C/I2S/IR (zdalne sterowanie),
- dostępne 10 GPIO,
- 1 wyprowadzenie ADC (0 – 3,3 V).

- **Konwerter USB-TTL (UART): CH340.**

- **Raster wyprowadzeń: 2,54 mm.**
- **Wymiary modułu: 58 × 30 mm.**

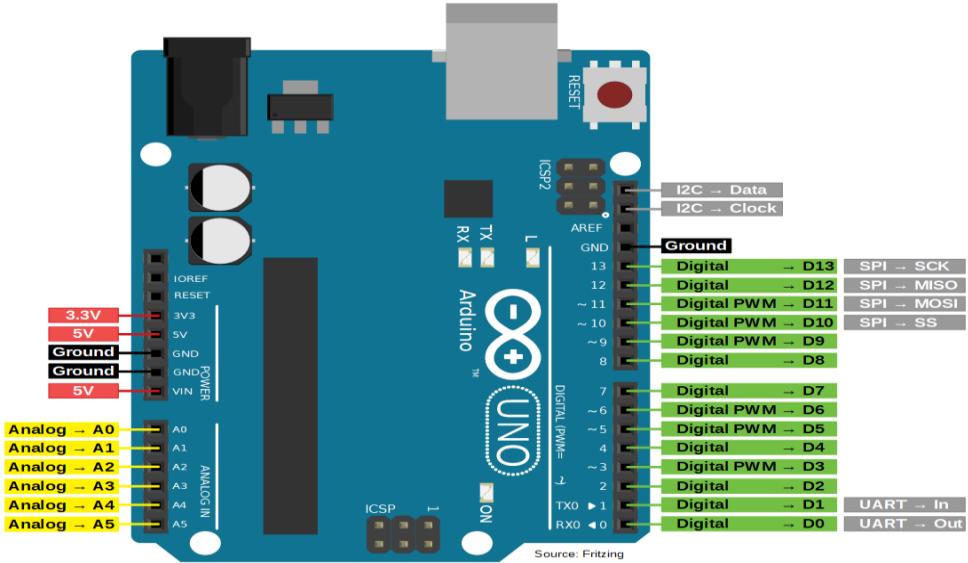
Cena: 24.90 zł



Rysunek 5. ESP8266 PinOut

Arduino Uno

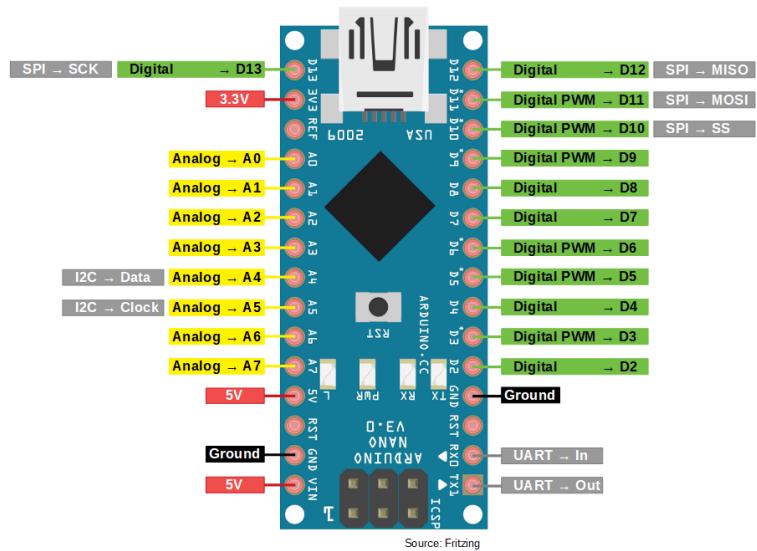
- 16 Mhz CPU
- 32 KiB pamięci flash
- 2 KiB SRAM
- 1 KiB EEPROM
- Ilość pinów I/O: 22
- Zasilanie: 7-12V
- Cena: 92.00 zł



Rysunek 6. Arduino Uno PinOut

Arduino Nano

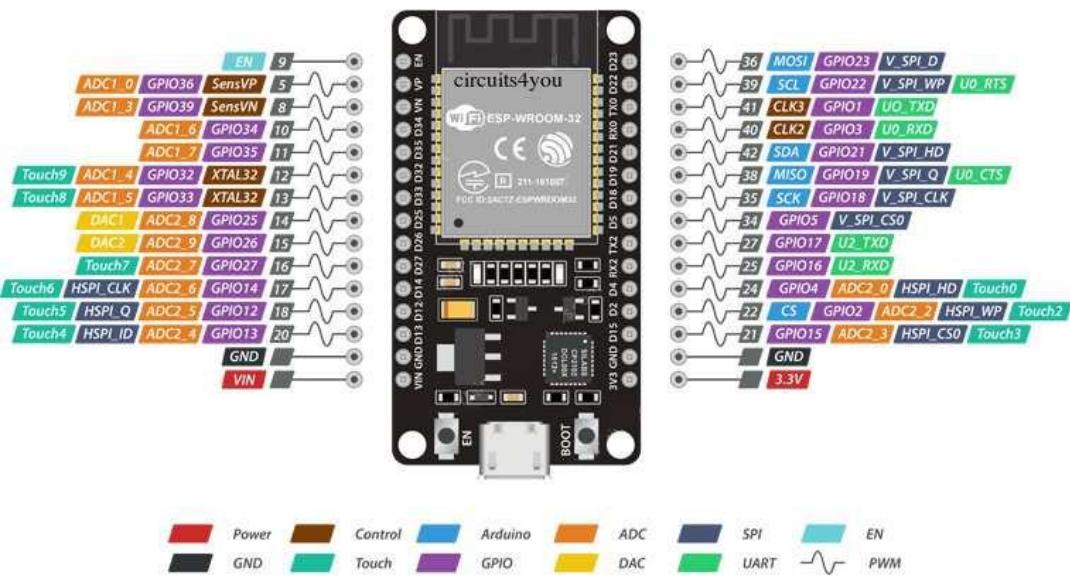
- 16 Mhz CPU
- 32 KiB pamięci flash
- 2 KiB SRAM
- 1 KiB EEPROM
- Ilość pinów I/O: 14
- Zasilanie: 7-12V
- Cena: 95.00 zł



Rysunek 7. Arduino Nano PinOut

ESP32

- Dual/Single Core pracujący z częstotliwością 160/240 MHz
- 520 KiB SRAM
- 448 KiB ROM
- Bluetooth v4.2 BR/EDR and BLE
- Wi-Fi 802.11 b/g/n
- Cena: 49.00 zł



ESP32 Dev. Board / Pinout

Rysunek 8. ESP32 PinOut

8.3. Pula czujników

DHT11

- Ogólne:

- Napięcie zasilania: 3 V do 5,5 V
- Pobór prądu: 0,2 mA
- Częstotliwość próbkowania: 1Hz

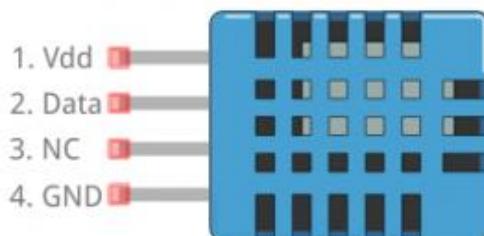
- **Wbudowany termometr**

- Zakres pomiarowy: 0 - 50 °C
- Dokładność: ±2°C

- **Czujnik wilgotności:**

- Zakres pomiarowy: 20 - 95%RH
- Dokładność: ±5%RH

- **Cena: 4.33 zł**



Rysunek 9. DHT11 PinOut

DHT22 (AM2302)

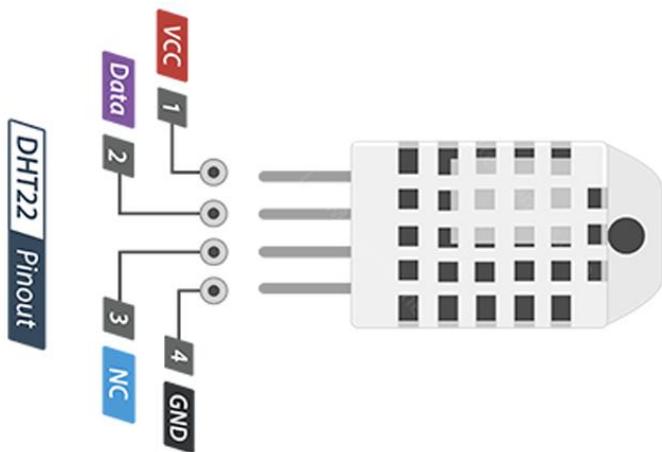
- **Napięcie zasilania: od 3,3 V do 6 V**
- **Średni pobór prądu: 0,2 mA**
- **Temperatura**

- Zakres pomiarowy: -40 do 80 °C
- Rozdzielcość: 8-bitów (0,1 °C)
- Dokładność: ± 0,5 °C
- Czas odpowiedzi: średnio 2 s

- **Wilgotność:**

- Zakres pomiarowy: 0 - 100 % RH
- Rozdzielcość: 8-bitów ($\pm 0,1 \%$ RH)
- Dokładność $\pm 2 \%$ RH*
- Czas odpowiedzi: średnio 2 s

- **Cena: 24.90 zł**



Rysunek 10. DHT22 PinOut

DHT21 (AM2301)

- **Model: DHT21 / AM2301**
- **Temperatura**
 - Zakres pomiarowy: od -40 do +80 °C
 - Rozdzielcość: 0,1 °C
 - Dokładność: +/- 0,2 °C
 - Czas odpowiedzi: 2 s
- **Wilgotność:**
 - Zakres pomiarowy: 0 - 100 %RH
 - Rozdzielcość: 0,1 % RH
 - Dokładność ± 1 RH (przy 25 °C)
 - Czas odpowiedzi: 2 s
- **Napięcie zasilania: 3,3 V - 5,5 V**
- **Pobór prądu: 1,5 mA**
- **Wymiary: 28 x 22 x 5 mm**
- **Cena: 24.38 zł**



Rysunek 11. DHT21 PinOut

8.4. Wybór kontrolera

Możliwości wszystkich kontrolerów są zbliżone, jednak tylko urządzenia ESP posiadają wbudowaną kartę WiFi. Ponieważ układy nie będą wymagać dużej mocy obliczeniowej, ani nie potrzebują dużo pamięci (RAM oraz flash), w tym przypadku wybór sprowadza się więc do porównania ceny między nimi.

Na potrzeby tego projektu wybraliśmy kontroler **ESP8266**.

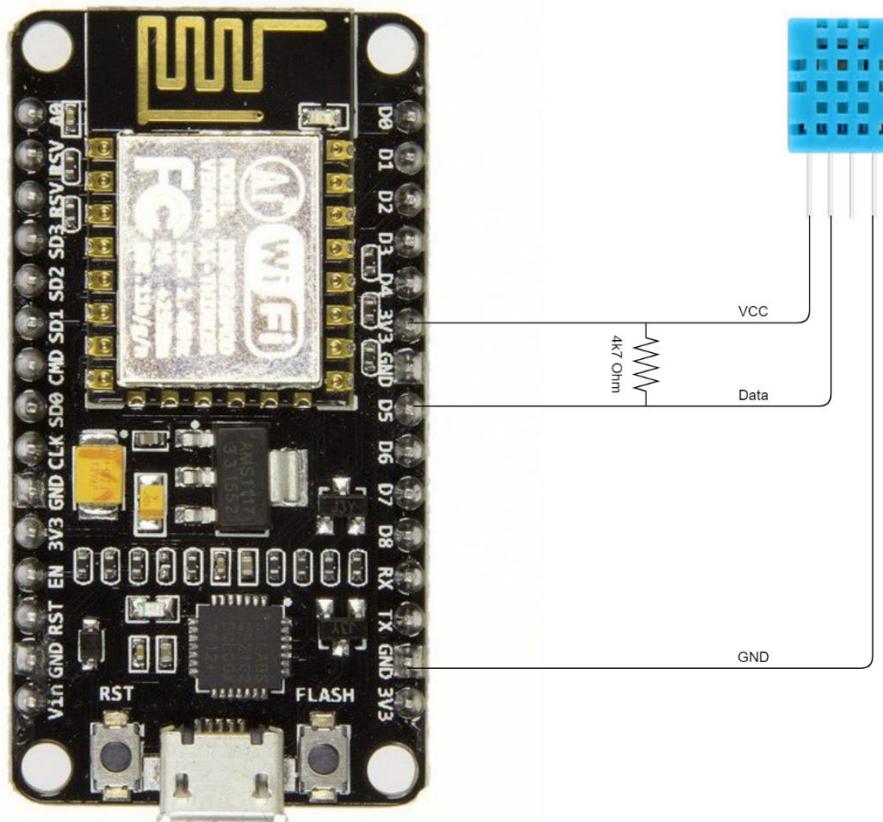
8.5. Wybór czujnika

Jedynie parametry które mogą rozróżnić te czujniki od siebie to zakres badanej wartości, jej dokładność oraz cena. Nie posiadając dokładnych wymagań klienta, a w szczególności takich mówiących o wyżej wymienionych czynnikach, uznaliśmy, że najbardziej kluczowym parametrem będzie cena.

Na potrzeby tego projektu wybraliśmy czujnik **DHT11**.

Jeżeli jednak pojawi się potrzeba zainstalowania bardziej dokładnego czujnika, wymiana na model DHT22 nie stanowi żadnego problemu. Jest to jedynie kwestia podłączenia go w ten sam sposób co czujnik DHT11.

8.6. Schemat elektryczny



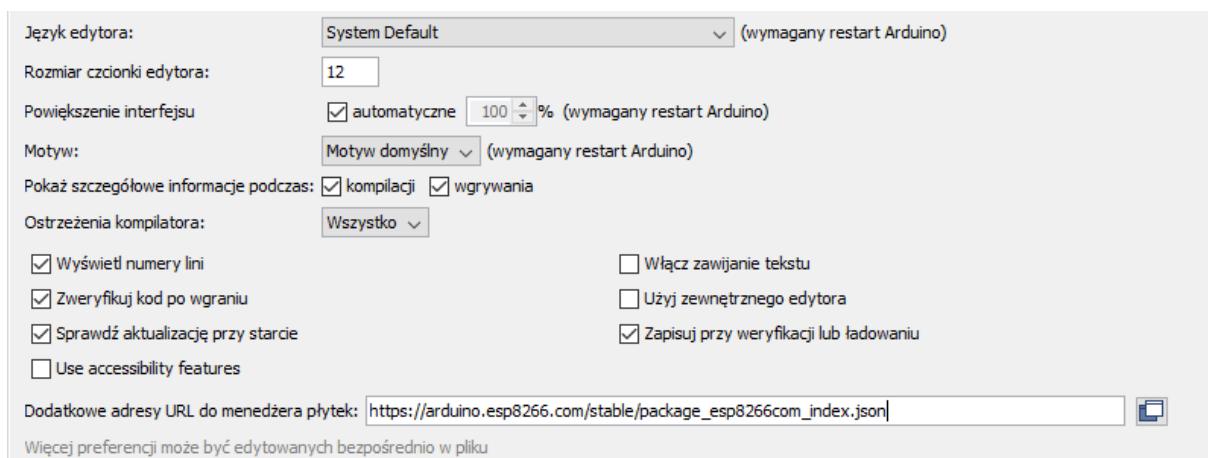
Rysunek 12. Schemat elektryczny

8.7. Sposób programowania

Programowanie ESP8266 przez Arduino IDE jest obecnie najprostszym i najbezpieczniejszym sposobem programowania tego kontrolera. Aby środowisko poprawnie rozpoznało inny niż kontroler niż Arduino należy pobrać pakiet bibliotek i informacji na temat wybranego przez nas urządzenia.

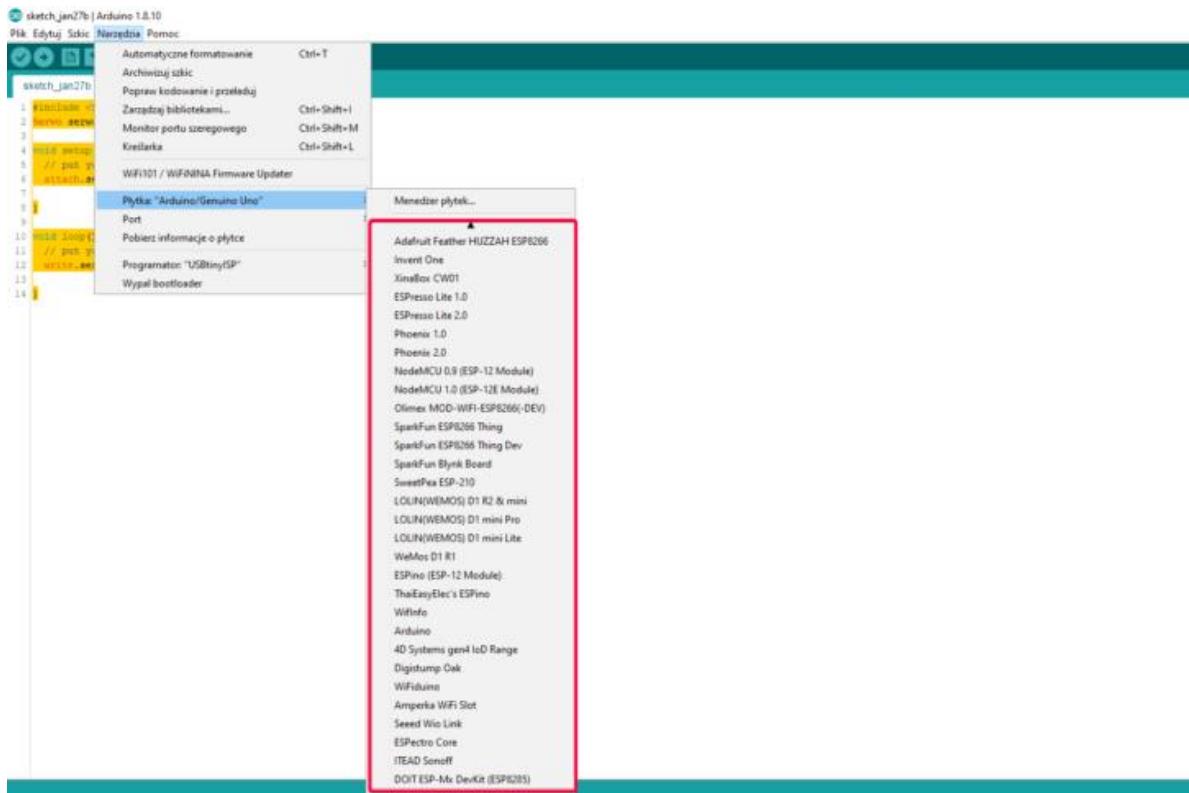
W Arduino IDE wybieramy opcję *Plik > Preferencje* i w polu *Dodatkowe adresy URL do menedżera płytek* wpisujemy poniższy adres:

https://arduino.esp8266.com/stable/package_esp8266com_index.json



Rysunek 13. Dodanie informacji o ESP8266 do Arduino IDE

W kolejnym kroku wybieramy opcję *Narzędzia > Płytki > Menedżer płytEK*, w wyszukiwarkę wpisujemy hasło "ESP8266" i instalujemy paczkę nazwaną "esp8266 by ESP8266 Community". Od tej pory podczas wyboru płytEK dostępne będą różne modele modułów z ESP8266 na pokładzie.



Rysunek 14. Wybór płytEK z ESP w Arduino IDE

8.8. Oficjalna dokumentacja

ESP8266: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

DHT11: <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>

9. Wybrane warstwy OSI

Wykonanie: Katarzyna Czajkowska

Sprawdzenie: Arkadiusz Cichy

9.1. Model OSI

7. Warstwa aplikacji	1. Warstwa fizyczna
6. Warstwa prezentacji	Warstwa aplikacji
5. Warstwa sesji	Warstwa transportowa
4. Warstwa transportowa	Warstwa Internetu
3. Warstwa sieci	Warstwa dostępu do sieci
2. Warstwa łączna danych	

Porównanie modelu odniesienia OSI z modelem protokołów TCP/IP.

TCP/IP jest modelem protokołów, określający dokładniej działanie zestawów protokołów w poszczególnej warstwie i pośredniczenie między siecią międzyludzką a siecią danych. OSI jest modelem odniesienia, pokazując jak poszczególne warstwy oddziałują ze sobą, jaka jest forma komunikacji między warstwami oraz zapewniając spójność między wszystkimi typami protokołów.

Warstwa dostępu do sieci z modelu TCP/IP reprezentowana jest pod postacią 2 warstw w modelu OSI, dodając fizyczny aspekt dostępu do sieci.

Warstwy 3 i 4 obu modeli są odpowiadające sobie, różniąc się jednak relacjami do innych warstw.

W modelu OSI warstwa aplikacji podzielona jest na warstwę sesji, warstwę prezentacji i warstwę aplikacji – są to zestawy protokołów odpowiedzialnych za funkcjonalność aplikacji dla użytkowników końcowych.

Dokładniejsze omówienie warstw modelu OSI, które będą dla nas istotne w projekcie. Kolejność malejąca (idąc kolejnością, jaką przechodzi strumień danych od aplikacji do przesłania do zdalnego hosta)

9.2. Warstwa aplikacji

7. Warstwa aplikacji
6. Warstwa prezentacji
5. Warstwa sesji
4. Warstwa transportowa
3. Warstwa sieci
2. Warstwa łącza danych
1. Warstwa fizyczna

protokoły warstwy aplikacji:

- DNS
- HTTP
- SMTP
- POP
- DHCP
- FTP
- IMAP

Warstwa 7 modelu OSI jest warstwą najbliższą użytkownikowi. Zapewnia interfejs pomiędzy aplikacjami a siecią.

Modele sieci:

- **P2P (peer-to-peer)** – bezpośrednie połączenie między dwoma urządzeniami końcowymi (peer). Urządzenia, połączone ze sobą przez sieć, mogą współdzielić zasoby oraz komunikować się bez pomocy osobnego serwera – każde urządzenie może być klientem albo serwerem.
- **klient-serwer** – role klienta i serwera są na stałe przypisane, urządzenie klienckie wysyła zapytanie o dane, na które serwer odpowiada wysyłając dane. Na tej zasadzie działa HTTP w naszym projekcie, gdzie urządzenie (komputer, telefon) wysyła żądanie do serwera, którym jest IoT, w celu uzyskania informacji z czujnika. Protokoły warstwy aplikacji opisują format żądań i odpowiedzi. Jest to również forma

bezpieczniejsza niż P2P, ponieważ mogą zostać nałożone ograniczenia uwierzytelnienia i identyfikacji typów danych.

Istotne protokoły:

- HTTP (Hypertext Transfer Protocol) – protokół przesyłania danych w sieci WWW, dokładniej opisany w punkcie 8,
- SMTP (Simple Mail Transfer Protocol), POP (Post Office Protocol) – protokoły obsługi poczty elektronicznej,
- DNS (Domain Name Service) – protokół zamieniający adres IP na nazwę domeny,
- DHCP (Dynamic Host Configuration Protocol) – protokół automatycznie przypisujący adresy IP,
- FTP (File Transfer Protocol) – protokół pobierania danych z serwera.

9.3. Warstwa transportowa

7. Warstwa aplikacji
6. Warstwa prezentacji
5. Warstwa sesji
4. Warstwa transportowa
3. Warstwa sieci
2. Warstwa łącza danych
1. Warstwa fizyczna

Protokoły warstwy transportowej:

- UDP
- TCP

Warstwa 4 modelu OSI odpowiada za nawiązanie sesji komunikacyjnej oraz wymianę danych między aplikacjami. Jest łącznikiem między aplikacją a warstwą sieci.

Główne zadania warstwy transportowej to:

- śledzenie indywidualnej komunikacji między aplikacjami na hoście źródłowym i docelowym – warstwa transportowa utrzymuje sesję między kilkoma aplikacjami na zdalnych hostach.
- segmentacja danych – łatwiejsze zarządzanie danymi, dzielenie ich na mniejsze części żeby dało się je wysłać w postaci pakietu. Warstwa transportowa zajmuje się przede wszystkim składaniem danych z segmentów w całość w celu przekazania ich do warstwy aplikacji.
- identyfikacja właściwej aplikacji dla każdego strumienia danych – na podstawie numeru portu warstwa transportowa identyfikuje usługę lub aplikację zawartą w strumieniu danych i przekazuje go tylko do właściwej aplikacji.

Dodatkowe informacje na temat protokołów warstwy 4 zostały przedstawione w punkcie 8.

9.4. Warstwa sieci

7. Warstwa aplikacji
6. Warstwa prezentacji
5. Warstwa sesji
4. Warstwa transportowa
3. Warstwa sieci
2. Warstwa łącza danych
1. Warstwa fizyczna

Protokoły warstwy sieci:

- IPv4
- IPv6

Warstwa 3 modelu OSI opisuje wymianę danych pomiędzy urządzeniami końcowymi przez sieć. Potrzebne do tego są:

- **adresacja urządzeń końcowych** – każde urządzenie końcowe ma przypisany swój adres IP
- **enkapsulacja** - datagramy PDU (Protocol Data Unit) otrzymane z warstwy transportowej zostają spakowane – dodawany jest nagłówek z informacjami o IP (adres nadawcy, adres odbiorcy)
- **routing** – wybieranie najlepszej ścieżki między nadawcą i odbiorcą. Proces realizowany jest przez router po przeanalizowaniu pakietu uzyskanego przez enkapsulację.
- **deenkapsulacja** – po otrzymaniu przez urządzenie docelowe, nagłówek pakietu sprawdzany jest w celu określenia, czy adres IP urządzenia zgadza się z adresem w nagłówku. Jeżeli urządzenie docelowe jest zamierzonym odbiorcą, pakiet zostaje rozpakowany i przekazany do warstwy transportowej.

Budowa nagłówka:

Najważniejsze elementy nagłówka pakietu IPv4 to:

- **wersja** – $0100_2 = 4_{10}$, wskazuje wersję protokołu IP (dla IPv6 będzie to $0110_2 = 6_{10}$)
- **DS (Differentiated Services)** – zróżnicowane usługi, dawniej typ usługi. Stosowane do określenia priorytetu pakietu.
- **TTL (Time To Live)** – czas życia (w skokach). Przyznany pakietowi przez nadawcę, wyznacza po ilu skokach (przetworzeniach przez router) pakiet zostanie odrzucony.
- **protokół** – typ danych przenoszonych w pakiecie. Wartość liczbową, na podstawie której warstwa sieci decyduje do jakiego protokołu przekazać dane. Przykłady formatu: dla ICMP (0x01), TCP (0x06), UDP (0x11)
- **suma kontrolna** – służy do sprawdzenia poprawności nagłówka. Wartość tego pola musi być identyczna z wyliczoną sumą kontrolną nagłówka.
- **źródłowy oraz docelowy adres IP** – dwa pola zawierające 32-bitowe wartości adresów IP nadawcy oraz odbiorcy.

9.5. Routing

Jeżeli zarówno host jak i odbiorca wyznaczeni w nagłówku pakietu znajdują się w tej samej sieci lokalnej (co stwierdzone jest przez porównanie adresów, znając maskę podsieci) przesłanie pakietu przebiega bezpośrednio między urządzeniami końcowymi. Jednak jeżeli odbiorca pakietu znajduje się w sieci zdalnej, pakiet ten wysyłany jest na adres **bramy domyślnej** sieci lokalnej, czyli adres interfejsu sieciowego routera podłączonego do sieci globalnej. Następnie pakiet ten jest przekazywany do innych routerów, znajdujących się w sieci zdalnej. Routing odbywa się na podstawie tablicy routingu, przechowywanej przez router. Trasy w tej tablicy mogą być skonfigurowane ręcznie lub automatycznie.

Wpis na tablicy routingu zawiera między innymi informacje o sieci docelowej, dystansie, adresie IP następnego skoku oraz interfejsie wyjściowym na routerze, prowadzącym do tej sieci.

10. Transmisja WiFi oraz TCP/IP

Wykonanie: Matusz Gurski

Sprawdzenie: Katarzyna Czajkowska

10.1. Transmisja WiFi

WiFi - Produkty bezprzewodowej sieci lokalnej oparte na standardach (IEEE) 802.11. Technologia ta umożliwia wielu urządzeniom bezprzewodową wymianę danych lub połączenie z internetem za pomocą fal radiowych.

Działa na podobnej zasadzie co inne urządzenia bezprzewodowe - wykorzystuje częstotliwości radiowe do wysyłania sygnałów między urządzeniami. Dane przekształcane są w sygnał radiowy i transmitowane a router bezprzewodowy odbiera go i dekoduje. Proces ten działa też w odwrotnym kierunku - gdy router przekształca dane na sygnał radiowy i transmituje a urządzenie docelowe odbiera sygnał i dekoduje go.

Sygnały nadawane są na częstotliwościach 2,4 GHz lub 5 GHz. Podstawowe różnice między tymi częstotliwościami to zasięg i szerokość pasma(prędkość). Częstotliwość 2,4 GHz zapewnia większy zasięg, ale przesyła dane z mniejszą prędkością. Częstotliwość 5 GHz zapewnia mniejszy zasięg, ale przesyła dane z większą prędkością.

Wyróżniamy wiele różnych standardów WiFi. Niektóre z nich to:

- **802.11a** Transmituje dane z częstotliwością 5 GHz. Zastosowane multipleksowanie z ortogonalnym podziałem częstotliwości (OFDM) poprawia odbiór, dzieląc sygnały radiowe na mniejsze sygnały przed dotarciem do routera. Maksymalna przepustowość do 54 Mb/s. Zasięg w zamkniętym pomieszczeniu przy maksymalnej prędkości - 10 m. Zasięg przy maksymalnej prędkości na świeżym powietrzu - 50 m.
- **802.11b** Transmituje dane na poziomie częstotliwości 2,4 GHz. Maksymalna przepustowość do 11 Mb/s. Zasięg w pomieszczeniu zamkniętym przy maksymalnej prędkości - 50 m. Zasięg na świeżym powietrzu przy maksymalnej prędkości - 100 m.
- **802.11g** Transmituje dane na poziomie częstotliwości 2,4 GHz. Może obsłużyć do 54 megabitów danych na sekundę. 802.11g jest szybszy, ponieważ podobnie jak 802.11a wykorzystuje on kodowanie OFDM. Zasięg w pomieszczeniu zamkniętym przy maksymalnej prędkości - 50 m. Zasięg na świeżym powietrzu przy maksymalnej prędkości - 100 m.
- **802.11n** Aktualnie najszerzej dostępny ze standardów. Wstecznie kompatybilny z 802.11a, b, g. Znaczco poprawił prędkość i zasięg w stosunku do swoich poprzedników. Maksymalna przepustowość do 300 Mb/s. Zasięg w pomieszczeniu zamkniętym przy maksymalnej prędkości - 110 m. Zasięg na świeżym powietrzu przy maksymalnej prędkości - 250 m.

Wybrany przez nas mikrokontroler posiada łączność WiFi w standardzie **802.11 b/g/n** co oznacza, że jest on kompatybilny ze standardami **802.11b**, **802.11g** i **802.11n**.

10.2. TCP/IP

TCP/IP to model, który pozwala na podział zagadnienia komunikacji sieciowej na szereg współpracujących ze sobą warstw.

Warstwy te to:

- Warstwa aplikacji - Warstwa w której pracują użyteczne dla człowieka aplikacje. Obejmuje zestaw gotowych protokołów takich jak HTTP, FTP, Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP) i Simple Network Management Protocol (SNMP).
- Warstwa transportowa - odpowiada za utrzymanie komunikacji typu end-to-end w sieci. TCP obsługuje komunikację między hostami i zapewnia kontrolę przepływu, multipleksowanie i niezawodność. Protokoły transportowe obejmują TCP i User Datagram Protocol (UDP), który czasami jest używany zamiast TCP do specjalnych celów.
- Warstwa internetu - Obsługa adresowania, pakowania i routingu.
- Warstwa dostępu do sieci - Zajmuje się przekazywaniem danych przez fizyczne połączenia między urządzeniami sieciowymi

Przedstawienie uproszczonego schematu działania modelu TCP/IP

- Po odebraniu danych w warstwie aplikacji, na przykład z przeglądarki internetowej, używając protokołu HTTP, warstwa aplikacji komunikuje się z warstwą transportową przez port, np. port 80 w przypadku protokołu HTTP i przekazuje dane.
- TCP - Warstwa transportowa dzieli dane na pakiety, które następnie wysłane będą w miejsce docelowe. TCP do każdego pakietu umieszcza również nagłówek, który jest instrukcją w jaki sposób z powrotem złożyć pakiety w całość.
- IP - Następnie w warstwie internetu, protokół IP "dołącza" do pakietów adres ip źródłowy - z którego pakiety zostały wysłane oraz adres ip docelowy - do którego pakiety zmierzą. Dane następnie przekazywane są dalej do warstwy ostatniej.
- Warstwa dostępu do sieci dba o adresowanie MAC, czyli o to by pakiety dotarły do odpowiedniego urządzenia fizycznego.

10.3. TCP a UDP

Protokoły TCP i UDP używane są do przesyłania danych.

Główne różnice między TCP a UDP:

- **TCP**
 - Protokół zorientowany połączeniowo. Po ustanowieniu połączenia dane mogą być przesyłane dwukierunkowo.
 - Nawiązywanie połączenia odbywa się przy pomocy procedury nazywanej three-way handshake.
 - Gwarantuje wysłanie wszystkich pakietów.
- **UDP**
 - Protokół bezpołączeniowy. Nie gwarantuje dostarczenia wszystkich pakietów. Szybszy od TCP.

10.4. HTTP

HTTP - Protokół bezstanowy, Serwer i klient nie przechowują informacji o wcześniejszych zapytaniach pomiędzy nimi oraz nie posiada stanu wewnętrznego. Każde kolejne zapytanie traktowane jest więc jako „nowe”.

Najczęściej spotykana jest komunikacja HTTP odbywającą się z wykorzystaniem protokołu TCP.

Komunikacja HTTP realizowana jest poprzez wysłanie żądania (request) do serwera, który następnie generuje odpowiedź (response).

Niektóre z metod HTTP:

- **GET** - służy do żądania danych z serwera. Metoda, która wykorzystywana będzie w naszej aplikacji, w której moduł HTTP zaimplementowany będzie jako klient i na żądanie użytkownika, czyli gdy ten zażąda pewnych danych np. aktualnej temperatury z jednego z czujników, wysyłany będzie request GET do serwera z prośbą o to by odesłał żądane dane.
- **POST** - służy do wysyłania danych do serwera w celu utworzenia / aktualizacji zasobu.
- **HEAD** - Metoda HEAD żąda odpowiedzi od serwera, podobnie jak metoda GET. Metoda HEAD jednak nie oczekuje treści(response body).

Format żądania HTTP (request) jest następujący:

- Linia określająca czasownik HTTP, zasób i wersje protokołu
- Linia zawierająca nagłówki
- Linia pusta, która oznacza koniec nagłówków
- Opcjonalnie - ciało wiadomości

Typowy nagłówek HTTP może w implementacji naszego projektu wyglądać następująco.

GET <http://192.168.4.1/temperature> HTTP/1.1

Format odpowiedzi HTTP (response) :

- Linia z wersją protokołu i statusem odpowiedzi - np. *HTTP/1.1 200 OK*
- Linia z nagłówkami
- Linia pusta, która oznacza koniec nagłówków
- Opcjonalnie - ciało wiadomości

10.5. MQTT

Lekki protokół transmisji danych. Przeznaczony jest do transmisji dla urządzeń niewymagających dużej przepustowości. Zapewnia prostą komunikację pomiędzy wieloma urządzeniami. Idealnie sprawdza się tam gdzie wymagana jest oszczędność przepustowości oraz energii a więc jest on idealny dla aplikacji IoT.

Klienci MQTT, łączą się z brokerem MQTT, który pełni role serwera.

Podstawowe koncepcje

- **Publish/Subscribe** - Klient po połączeniu się z brokerem, może subskrybować dany temat lub publikować informacje w danym temacie.
- **Messages** - Informacje wymieniane pomiędzy urządzeniami - dane lub komendy
- **Topics** - Tematy do których subskrybują klienci by otrzymywać z nich informacje, lub do których publikują oni informacje.
- **Broker** - Odpowiedzialny za odbieranie wiadomości, filtrowanie ich, oraz publikowania wiadomości do subskrybujących dany temat klientów.

11. Podział na podsieci

Wykonanie: Katarzyna Czajkowska

Sprawdzenie: Arkadiusz Cichy

Do komunikacji z urządzeniami IoT potrzebne jest urządzenie sieciowe. Do uzyskania połączenia między urządzeniami potrzebny jest router lub switch. Router umożliwia połączenie zewnętrzne do Internetu, w momencie, gdy switch wyłącznie służy do lokalnych połączeń. Cenowo switch wychodzi bardziej opłacalnie, jednak nie spełnia całkowicie naszych zapotrzebowan. Wprawdzie nie jest nam potrzebne połączenie zewnętrzne, ponieważ urządzenia nie wychodzą poza lokalną komunikację, jednak łączność bezprzewodowa jest w naszym projekcie istotnym aspektem, czego switch nie umożliwia. Większość routerów umożliwia komunikację przez Wi-Fi oraz daje ograniczoną możliwość połączenia przez kabel, co przyda się jako awaryjny sposób połączenia urządzeń. Switche skupiają się na fizycznym aspekcie połączeń.

11.1. Teoria podsieci

Mając jedną sieć lokalną, można podzielić ją na podsieci. Jest to przydatne na przykład jeżeli chcemy ograniczyć dostęp do konkretnych urządzeń lub zasobów. Dzielenie na podsieci odbywa się przez konkretne przypisanie adresów IP oraz maski podsieci.

Każda podsieć musi mieć odpowiednio ustawioną adresację w celu umożliwienia komunikacji między zawartymi w niej urządzeniami. Żeby urządzenia mogły komunikować się bezpośrednio, adresy muszą być w tej samej podsieci, czyli adresy hostów muszą być dobrane z odpowiedniej puli wyznaczonej przez maskę podsieci oraz adres sieci.

Struktura adresu IP – cztery bajty oddzielone kropkami, np. 192.168.0.1

Maska podsieci wyznacza ile bitów adresu IP przeznaczone jest na identyfikację sieci, a ile na adres hostów. Im więcej bitów przeznaczonych na adres sieci, tym mniejsza pula adresów hostów, z których można w danej sieci skorzystać. Każda sieć musi mieć swój adres sieci (pierwszy możliwy adres) oraz adres rozgłoszeniowy (ostatni możliwy adres). Oznacza to, że ilość możliwych hostów w sieci to $2^n - 2$, gdzie n oznacza ilość bitów w części hosta.

Przykład:

Adres IP 192.168.0.44

Maska podsieci 255.255.255.0

Można to również zapisać jako 192.168.0.44/24, zapisując maskę od razu za adresem IP. Przydział bitów na adres sieci to 24, zostawiając 8 bitów na adres hosta. Daje to $2^8 - 2$ możliwych adresów hostów.

Adres sieci: 192.168.0.0

Adres rozgłoszeniowy: 192.168.0.255

Zakres adresów możliwych do wykorzystania: od 192.168.0.1 do 192.168.0.254

Przy klasycznym podziale na podsieci maska podsieci jest stała dla każdej podsieci. Jeżeli występuje konieczność oszczędzania adresów IP, podział na podsieci może również odbywać się z VLSM (Variable Length Subnet Masking) – zmienną długością maski podsieci. Każda podsieć może mieć inną maskę, pozwalając na tworzenie 2-hostowych podsieci dla połączeń między urządzeniami, 30-hostowych oraz 254-hostowych podsieci w tej samej sieci. Dla większości sieci domowych lub w małych firmach jest to jednak całkowicie zbędne, ponieważ pula dostępnych adresów wewnętrznych przypadających na jeden adres zewnętrzny pozwala na swobodne rozporządzanie adresami podsieci dla łatwiejszej adresacji.

11.2. Adresacja

W sieci, w której znajdująć się będzie nasz projekt, mogą być podłączone inne urządzenia poza naszymi czujnikami oraz komputerem/telefonem z którego się łączymy. Żeby uniknąć interferencji, urządzenia (komputer/telefon, urządzenia IoT) dodane przez nasz projekt powinny znajdować się w osobnej podsieci.

Klasyczną pulą adresów lokalnych są adresy 192.168.0.0 - 192.168.255.255 (dostępne są jeszcze inne warianty prywatnych adresów IP dające większą pulę, jednak 65 tysięcy dostępnych adresów zdecydowanie wystarczy w naszym przypadku).

Najprostszym podejściem jest przyjęcie maski 255.255.255.0 (zapisywane także jako /24) dającej 254 hostów na podsieć. Nie zakładamy więcej niż 80 urządzeń IoT, więc można przyjąć taką maskę.

Przykładowo wybrana podsieć może wyglądać w następujący sposób:

adres podsieci: 192.168.100.0

adresy hostów: 192.168.100.1 – 192.168.100.254

adres broadcastowy: 192.168.100.255

Adres podsieci może być inny, z puli 256 adresów uzyskanych przez zmienianie trzeciej liczby adresu IP (192.168.0.0, 192.168.1.0, ..., 192.168.255.0), jednak przy dołączaniu do istniejącej sieci należy uważać, żeby w adresach hostów wybranego adresu podsieci nie znajdowały się inne urządzenia, doprowadzając do konfliktu adresów.

Po wybraniu adresu podsieci pierwszy dostępny adres hosta (192.168.100.1) można przypisać do komputera/telefonu który będzie się komunikował z czujnikami, następnie w celu zachowania przejrzystej adresacji, urządzeniom IoT można przypisać adresy zaczynając od 192.169.100.11, traktując adres jako rodzaj identyfikacji, traktując to jako oznaczenie np. pierwszego czujnika z pierwszej grupy czujników. Pula adresów w podsieci o masce /24 zostawia zapas umożliwiający systematyczne nazewnictwo.

12. Implementacja obsługi protokołu HTTP w aplikacji mobilnej

Wykonanie: Mateusz Gurski

Sprawdzenie: Arkadiusz Cichy

Obsługa protokołu HTTP w aplikacji mobilnej zrealizowana została przy użyciu zewnętrznej biblioteki OkHttp, dostępnej pod adresem:

<https://square.github.io/okhttp/>

Implementacja wysyłania zapytań http przy pomocy biblioteki OkHttp prezentuje się następująco:

```
OkHttpClient okHttpClient = new OkHttpClient();  
  
Request request = new Request.Builder()
```

```

        .url(url)
        .build();

okHttpClient.newCall(request).enqueue(new Callback() {
    @Override
    public void onFailure(@NotNull Call call, @NotNull IOException e) {
        final String myResponse = e.getMessage().toString();
        MainActivity.this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                outText.append("\nOdpowiedź: ");
                outText.append(myResponse);
            }
        });
    }

    @Override
    public void onResponse(@NotNull Call call, @NotNull Response response) throws
IOException {
        if(response.isSuccessful()) {
            final String myResponse = response.body().string();
            MainActivity.this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    outText.append("\nOdpowiedź: ");
                    outText.append(myResponse);
                }
            });
        }
    }
})
}

```

12.1. Test obsługi protokołu HTTP w aplikacji mobilnej

Po połączeniu się z dostępnym urządzeniem, wybraniu zakładki sensory i naciśnięciu przycisku Odczyt, otrzymujemy odczyty z dostępnych przez dane urządzenie czujników.

Id urządzenia	Nazwa	Odczyt
1	test	23.20C
1	test	57.00%
1	test	57.00%
1	test	23.20C
1	test	23.20C
1	test	57.00%
1	test	23.20C
1	test	53.00%

ODCZYT

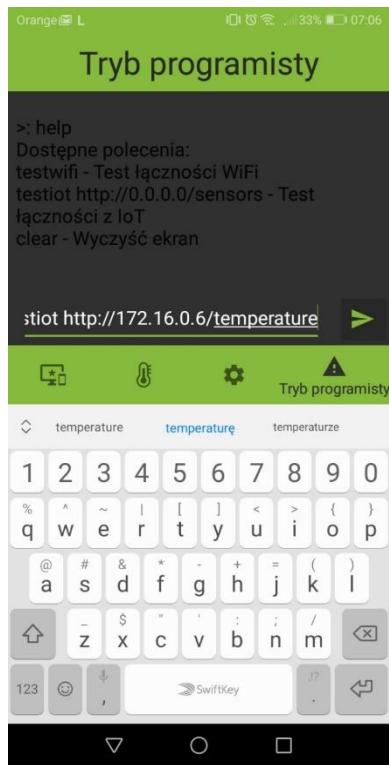
Urządzenia Sensory Ustawienia

◀ ○ □

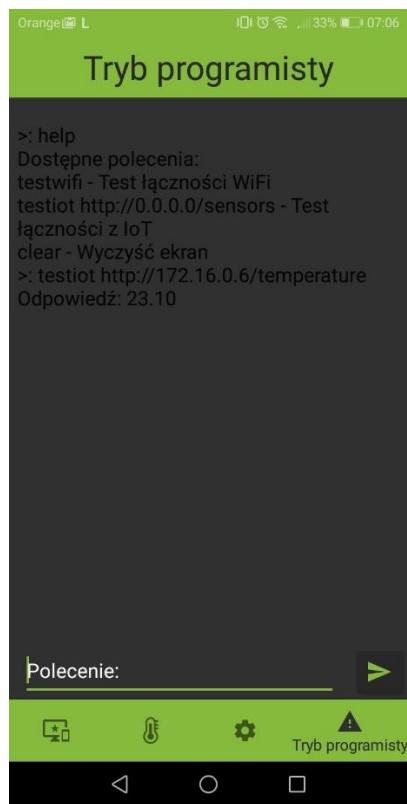
Rysunek 15. Odczyt danych z sensorów

12.2. Test obsługi protokołu HTTP w aplikacji mobilnej przy wykorzystaniu trybu administratora

W razie problemów, można również skorzystać z trybu administratora, który do wysyłania zapytań http, również wykorzystuje bibliotekę OkHttp. Tryb administratora pozwala między innymi na wysyłanie zapytań bezpośrednio na adres urządzenia IoT z pominięciem logiki aplikacji.

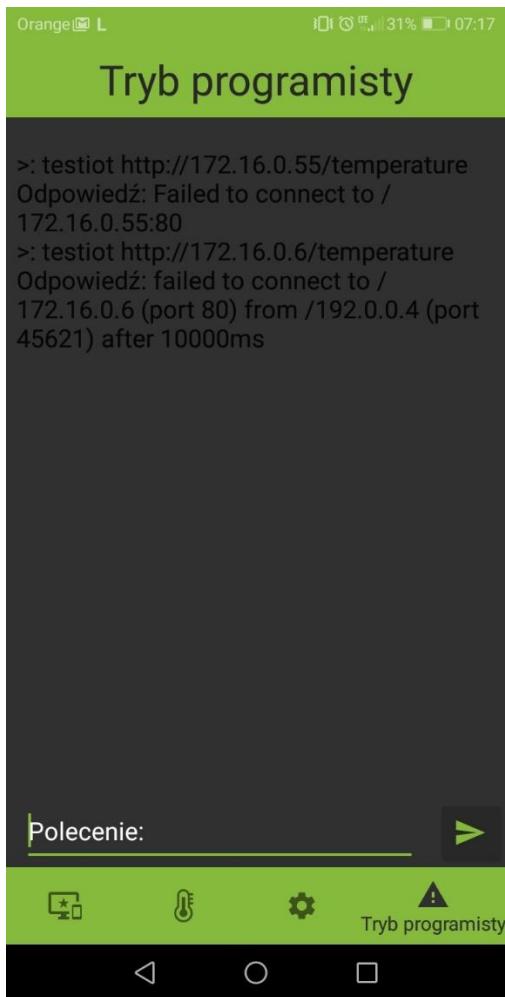


Rysunek 16. Test HTTP w aplikacji



Rysunek 17. Odpowiedź IoT

W przypadku wystąpienia błędów otrzymamy stosowną odpowiedź:



Rysunek 18. Informacja o błędzie połączenia

13. Implementacja obsługi protokołu HTTP w aplikacji desktopowej

Wykonanie: Mateusz Gurski

Sprawdzenie: Arkadiusz Cichy

Obsługa protokołu HTTP zrealizowana została jako osobna biblioteka, która dołączana jest do głównej aplikacji. Realizacja kolejnych protokołów jako osobne biblioteki pozwala na niezależną od głównej aplikacji ich realizację i umożliwia rozbudowę aplikacji o kolejne protokoły bez potrzeby wprowadzania dużych zmian do jej kodu. Moduł HTTP realizuje wysyłanie zapytań GET do serwera. Wykorzystane zostały klasy QNetworkAccessManager, QNetworkRequest i QNetworkReply dostarczane przez framework Qt.

UWAGA: Terminy używane w obecnej części dokumentu Sygnały oraz Sloty odnoszą się do funkcjonalności oferowanej przez framework Qt i nie są częścią protokołu HTTP.

Sygnały – pozwalają na komunikację technicznej części aplikacji z interfejsem bez zakłócania pracy interfejsu. Emitowane są przez obiekt gdy jego wewnętrzny stan ulega zmianie lub gdy zajdzie zmiana, która może zainteresować właściciela obiektu.

Slot – Są wywoływanie gdy sygnał do którego są podłączone zostanie wysłany. Sloty są normalnymi funkcjami C++ z jedyną różnicą, że można do nich podłączyć sygnały.

13.1. Omówienie podstawowych funkcji modułu realizującego obsługę protokołu HTTP

Podstawową funkcją biblioteki HTTP jest funkcja `get_request`, która pozwala na wysłanie zapytania GET na określony adres.

```
void Http_client::get_request(QString location)
{
    const QUrl url = QUrl(location);

    QNetworkRequest request(url);

    manager->get(request);
}
```

Po wysłaniu zapytania, oczekiwana jest odpowiedź serwera.

Wykorzystanie slotów i sygnałów pozwala na uruchomienie funkcji `readReady` po wyemitowaniu sygnału `finished` przez managera - w momencie w którym zakończy on odbiór odpowiedzi od serwera. Połączenie sygnału `finished` ze slotem `readReady` odbywa się w konstruktorze.

```
Http_client::Http_client(QObject *parent) : QObject(parent)
{
    connect(manager, SIGNAL(finished(QNetworkReply*)), this,
            SLOT(readReady(QNetworkReply*)));
}
```

Slot `readReady`, uruchamiany po otrzymaniu odpowiedzi prezentuje się następująco. Odpowiedź jest wyczytywana a następnie emitowana jako sygnał `dataReady`, który odbierany jest w głównej aplikacji.

```
void Http_client::readReady(QNetworkReply *reply)
{
    QByteArray data = reply->readAll();

    emit(dataReady(data));
}
```

By odbierać dane z utworzonego modułu HTTP, wcześniej omówiony sygnał `dataReady(data)` musi zostać połączony ze slotem w aplikacji odpowiedzialnym za odbieranie odpowiedzi i jej dalsze przetwarzanie. W tym celu, w aplikacji utworzony został odpowiedni `connect`, jak widać łączy on sygnał `dataReady(data)` należący do obiektu `http` ze slotem `http_get_response(data)` należącym do głównej aplikacji.

...

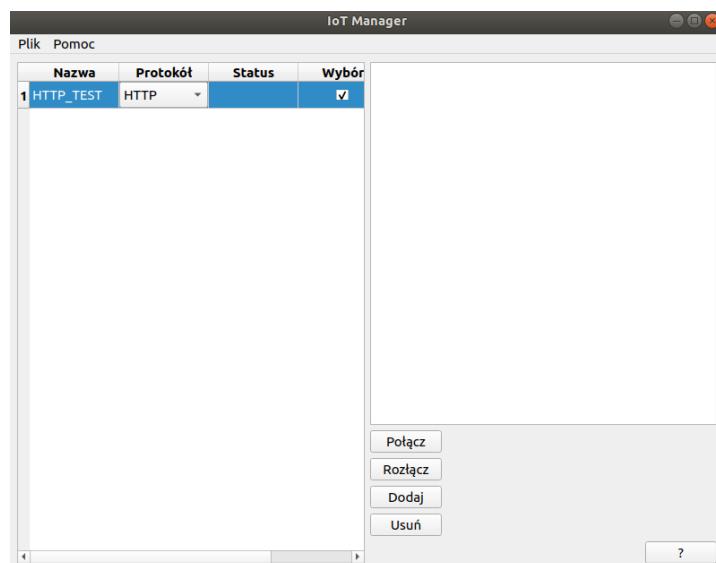
```
connect(&http, SIGNAL(dataReady(QByteArray)), this,
        SLOT(http_get_response(QByteArray)));
```

...

13.2. Test obsługi protokołu http w aplikacji desktopowej

Obecny stan aplikacji desktopowej oraz urządzenia IoT pozwala już na przeprowadzenie testu utworzonego modułu HTTP w aplikacji.

Po dołączeniu do aplikacji plików *.so i plików nagłówkowych modułu http, przycisk Połącz należący do interfejsu graficznego aplikacji został oprogramowany do łączenia się z wybranym urządzeniem IoT. Wybieramy urządzenie i odpowiedni protokół w następujący sposób.

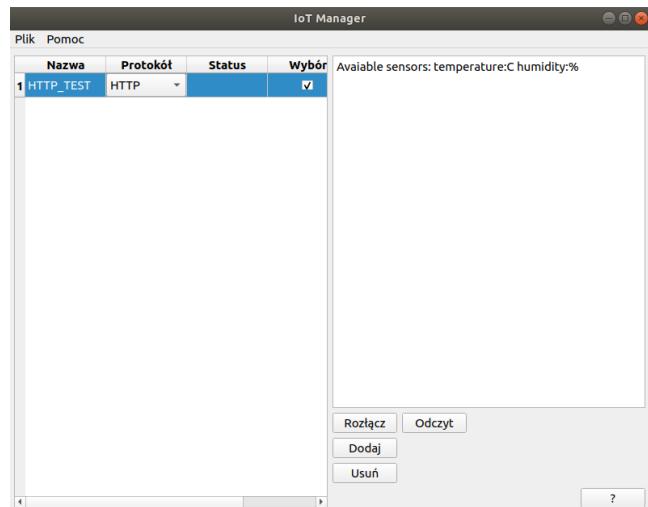


Rysunek 19. Wybór urządzenia

Funkcja wywoływana po naciśnięciu przycisku Połącz przedstawiona została poniżej. Jeśli zostało wybrane urządzenie, sprawdzany jest obsługiwany przez to urządzenie protokół, jeśli jest to HTTP, wysyłane jest zapytanie get na adres IP wybranego urządzenia z zapytaniem o dostępne przez dane urządzenie czujniki.

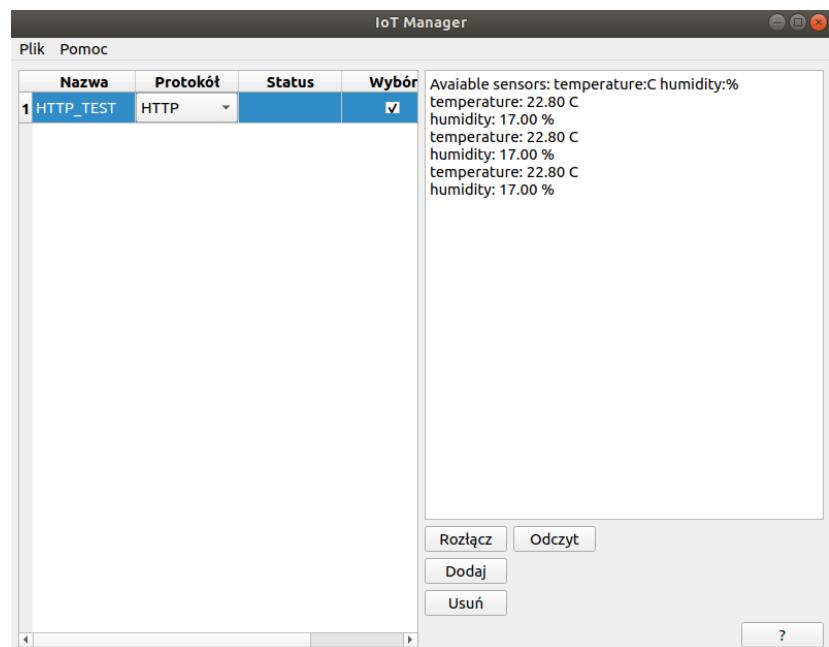
```
void UserInterface::on_pushButton_clicked()
{
    QModelIndexList selection = ui->iot_table->selectionModel()->selectedRows();
    if (selection.size() > 0) {
        QModelIndex index = selection.at(0);
        int row = index.row();
        if (devices[row]->getProtocol() == 1)
            connected_device = devices[row];
        http.get_request("http://" + connected_device->getIP() + "/sensors");
        ui->pushButton->setVisible(false);
    }
}
```

Wyświetlana jest odpowiedź z serwera HTTP. Przycisk Połącz „chowa się” i udostępniony zostaje przycisk Odczyt.



Rysunek 20. Odczyt danych z sensorów

Przycisk Odczyt pozwala na odczyt danych z dostępnych czujników i wyświetlanie ich. Naciśnięcie przycisku Rozłącz czyści ekran i powoduje „rozłączenie” z aktualnym urządzeniem.



Rysunek 21. Zakończenie połączenia

Przycisk odczyt sprawdza obsługiwany przez „połączone” urządzenie protokoły, jeśli jest to HTTP, w pętli wysyłane są kolejne zapytania na dostępne przez to urządzenie czujniki. Lista dostępnych przez urządzenie czujników tworzona jest w slocie http_get_response po nawiązaniu „połączenia”.

```
void UserInterface::on_pushButton_3_clicked()
{
    if (connected_device != nullptr) {
        if (connected_device->getProtocol() == 1) {
            for (int i=0;i<sensors_list.size();i++) {
                http.get_request("http://" + connected_device-
>getIP() + "/" + sensors_list[i]);
            }
        }
    }
}
```

```
}
```

Wysyłanie zapytań http związane jest ze slotem `http_get_response` co opisane zostało w punkcie „Implementacja obsługi protokołu HTTP w aplikacji desktopowej”. Slot ten odpowiedzialny jest za przetwarzanie otrzymanych odpowiedzi i wyświetlanie ich w oknie interfejsu graficznego.

```
void UserInterface::http_get_response(QByteArray data)
{
    QString DataAsString = QString(data);
    if(ui->pushButton_3->isVisible()){
        sensor_readings.append(data);
        if(sensor_readings.size() == sensor_units.size() &&
sensor_units.size()>0){
            for(int i = 0;i<sensor_readings.size();i++) {
                QString reading = sensors_list[i] + ":" + sensor_readings[i] + "
" + sensor_units[i];
                ui->textEdit->append(reading);
            }
            sensor_readings.clear();
        }
    }
    else{
        ui->textEdit->clear();
        ui->textEdit->setText("Available sensors: " + data);
        sensors = data;
        QStringList sensors_tmp_list = sensors.split(" ");
        sensor_units.clear();
        for(int i=0;i<sensors_tmp_list.size();i++){
            QStringList tmp = sensors_tmp_list[i].split(":");
            if(tmp.size() > 1){
                sensors_list.append(tmp[0]);
                sensor_units.append(tmp[1]);
            }
        }
        ui->pushButton_3->setVisible(true);
    }
}
```

14. Implementacja obsługi protokołu MQTT w aplikacji desktopowej

Wykonanie: Arkadiusz Cichy

Sprawdzenie: Szymon Cichy

14.1. Obsługa protokołu MQTT

Do obsługi protokołu przez aplikację jest zbudowana przez nas osobna biblioteka. Umożliwia ona zarówno subskrypcję jak i publikowanie informacji. Dzięki temu biblioteka może być stosowana do programu sterującego czujnikiem oraz aplikacji z której korzysta użytkownik. Inne funkcje tej biblioteki to: nawiązywanie połączenia, aktualizowanie logu, obsługa GUI.

14.2. Kluczowe funkcje biblioteki

Podstawową metodą jest `ToggleConnection()`. Zależnie od stanu, połączenie jest nawiązywane lub zamkane.

```
void MqttLib::ToggleConnection()
{
    if(m_client->state() == QMqttClient::Disconnected) {
        m_client->connectToHost();
    } else {
        m_client->disconnectFromHost();
    }
}
```

```
}
```

Kolejną funkcjonalnością jest publikowanie informacji za pomocą Publish. Jeśli informacja zostanie poprawnie wysłana, zwraca wartość true. W przeciwnym wypadku zwraca wartość false.

```
bool MqttLib::Publish(QMqttTopicName &name, QByteArray &message)
{
    if (m_client->publish(name, message) == -1)
        return false;
    return true;
}
```

Subskrypcja informacji realizowana jest za pomocą subscribe. Wyjściem funkcji jest wskaźnik na obiekt subskrybowany.

```
QMqttSubscription* MqttLib::Subscribe(QMqttTopicFilter &filter)
{
    auto subscription = m_client->subscribe(filter);
    if (!subscription)
        return nullptr;
    return subscription;
}
```

14.3. Pozostałe funkcje biblioteki

SetClientPort pozwala na zmianę portu.

```
void MqttLib::SetClientPort(int p)
{
    m_client->setPort(p);
}
```

Konstruktor tworzy obiekt QMqttClient do obsługi wszystkiego związanego z przesyłaniem danych za pomocą tego protokołu.

```
MqttLib::MqttLib()
{
    m_client = new QMqttClient();
}
```

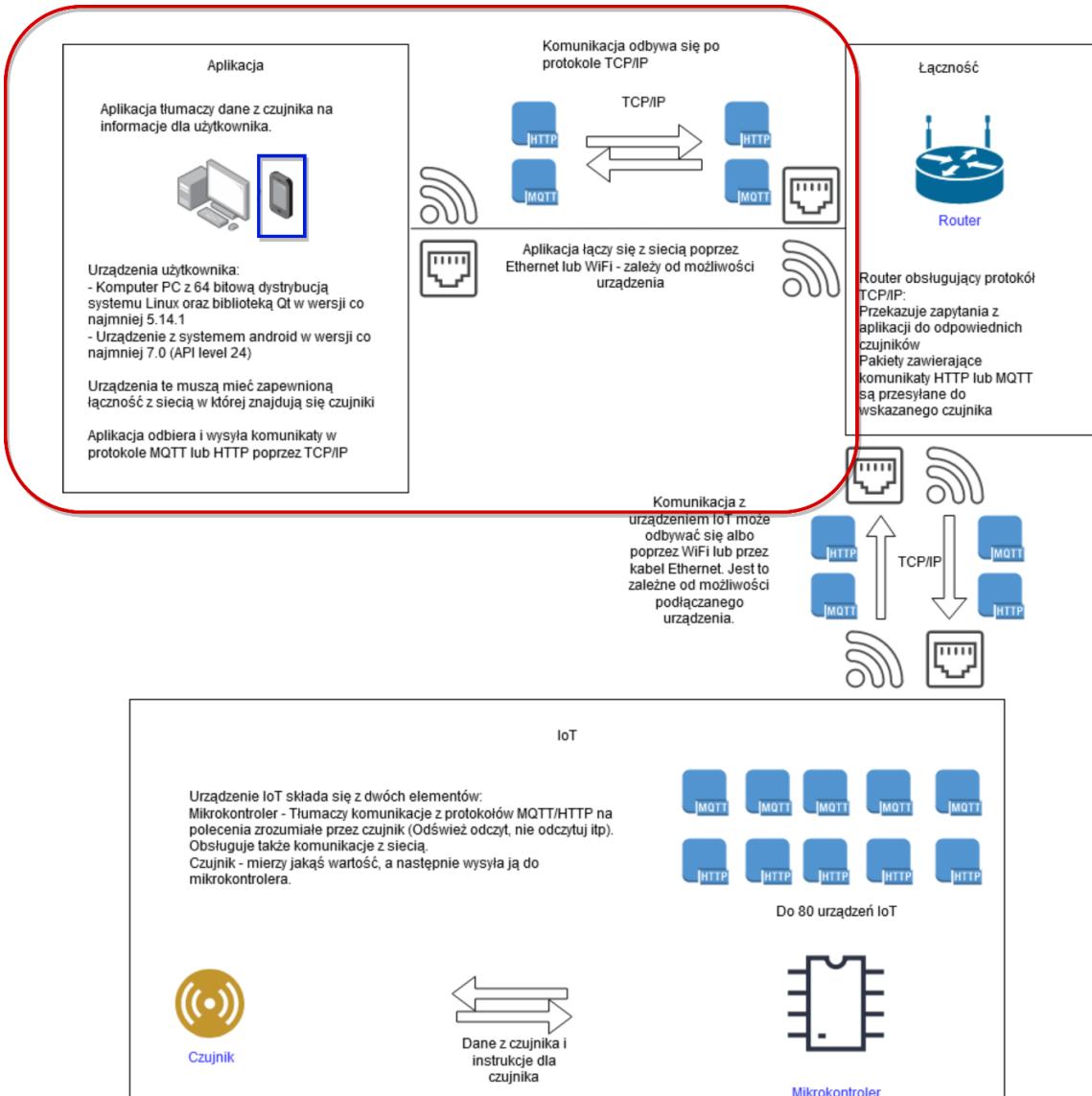
15. Oprogramowanie techniczne WiFi (Aplikacja użytkownika na system Android)

Wykonał: Mateusz Gurski/Adam Krizar

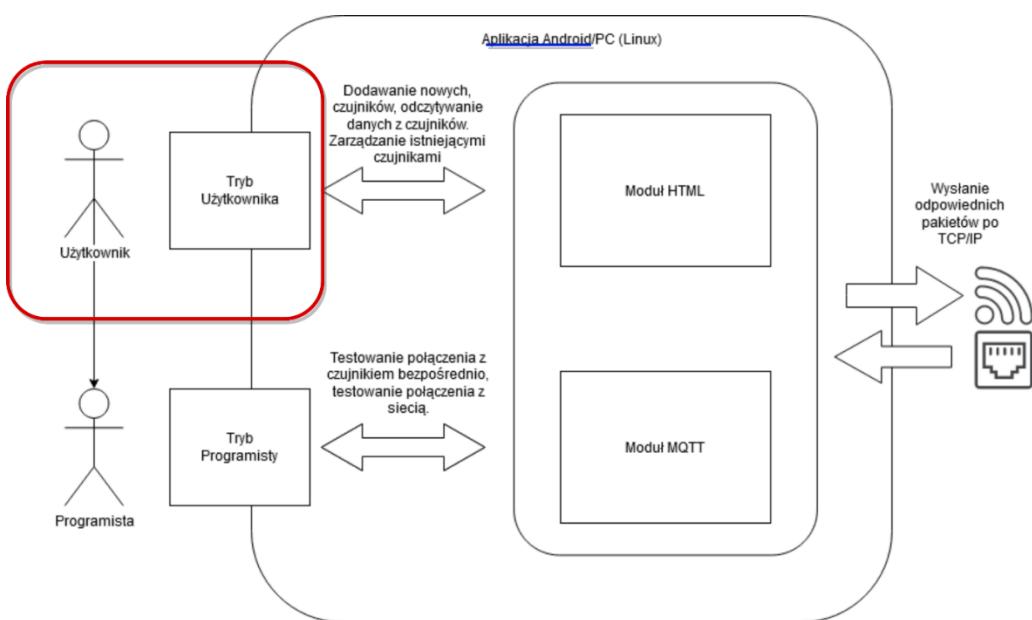
Sprawdzenie: Katarzyna Czajkowska

15.1. Przeznaczenie

Tryb użytkownika w aplikacji na system Android, pozwala na dodawanie nowych czujników, zarządzanie istniejącymi i wyświetlanie odczytów z czujników wykorzystując protokół komunikacji HTTP lub MQTT. Rola tego oprogramowania w schemacie ogólnym przedstawiona została poniżej.



Rysunek 22. Rola aplikacji na system Android w schemacie ogólnym



Rysunek 23. Tryb użytkownika aplikacji na system Android

15.2. Uruchomienie

15.2.1. Środowisko Programistyczne

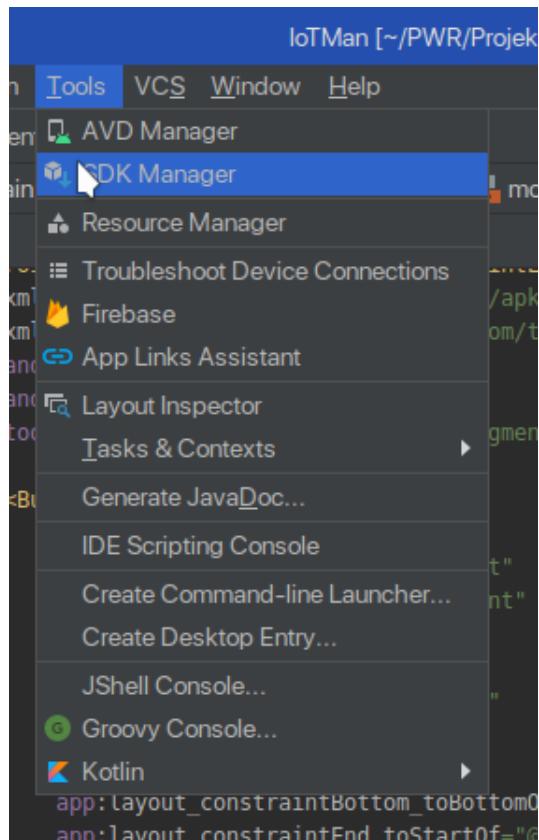
Aplikacja jest tworzona z wykorzystaniem programu Android Studio w wersji 3.6.2. Jest to narzędzie dedykowane do tworzenia aplikacji na urządzenia z systemem Android.



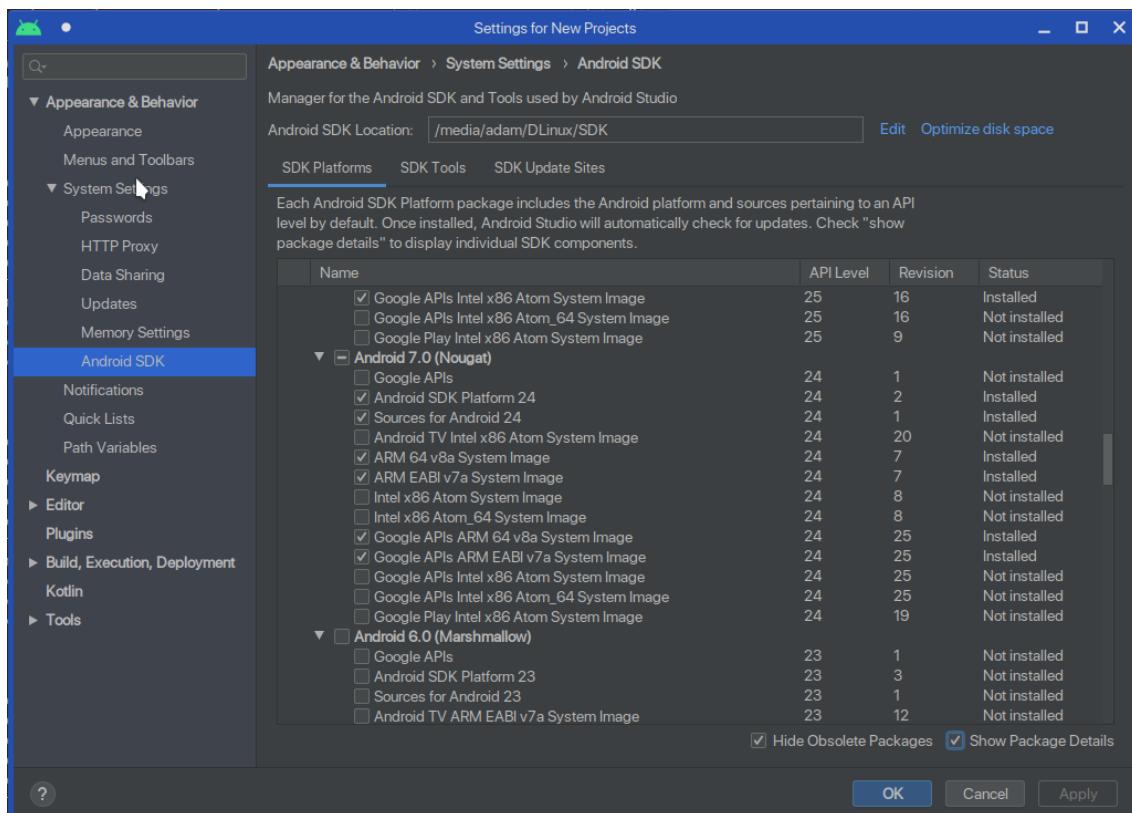
Rysunek 24. Informacje o Android Studio

Jako docelową wersję systemu android wybraliśmy systemy nowsze od androida 7 włącznie (API 24). Gwarantuje to, że aplikacja zadziała na większości smartfonów i tabletów na tą platformę – ponad 80% urządzeń. Wybrana przez nas wersja posiada także łatki bezpieczeństwa zeszłego roku co zapewnia minimalny poziom bezpieczeństwa dla naszej aplikacji.

Aby zaopatrzyć się w odpowiednią wersję SDK do tworzenia aplikacji najlepiej jest skorzystać z wbudowanego w program Android Studio menadżera. Pobierz on automatycznie wszystkie elementy potrzebne użytkowania programu z wybraną SDK.



Rysunek 25. Dostęp do menedżera SDK



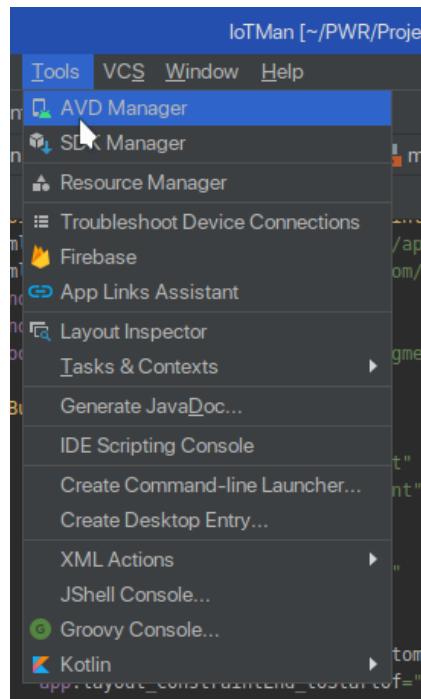
Rysunek 26. Wybór SDK

15.2.2. Uruchamianie aplikacji

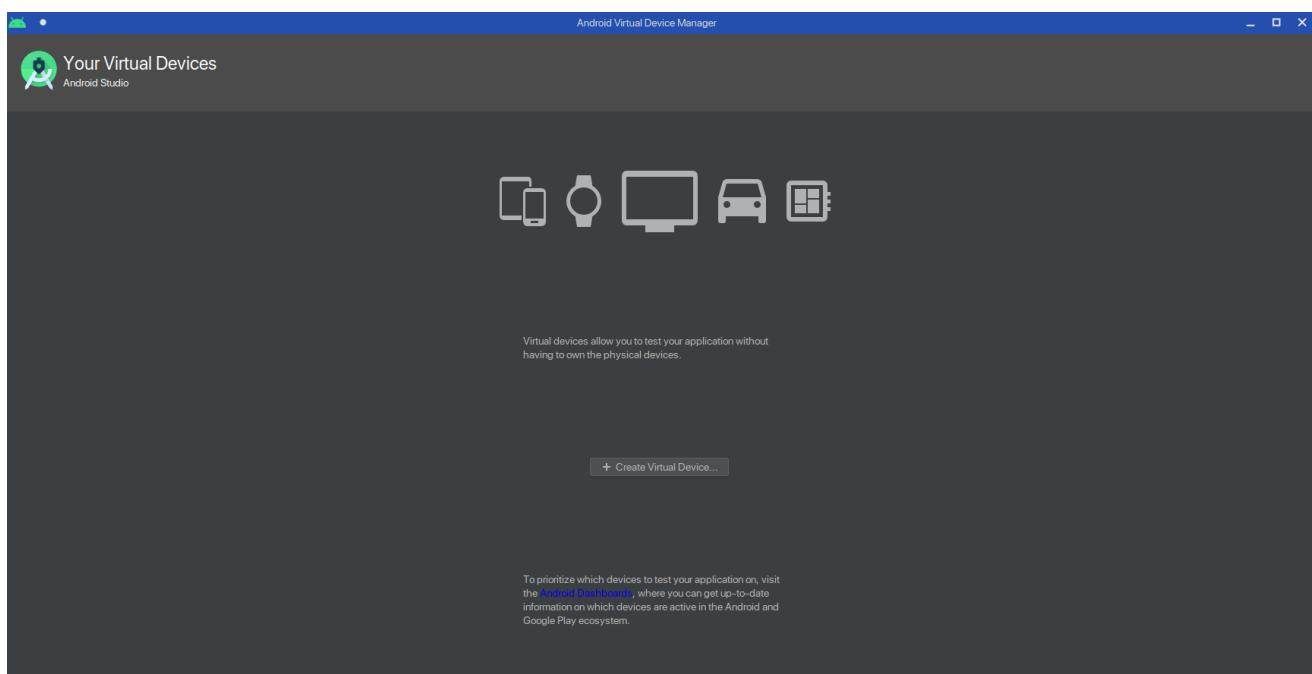
Program może być przetestowany na istniejącym urządzeniu z systemem android w wersji co najmniej 7.0 (Nougat). Wystarczy w takim przypadku skopiować załączony plik .apk do pamięci urządzenia, zezwolić

na instalowanie z nieznanych źródeł i uruchomić. Możliwe jest także wykorzystanie wbudowanej w program Android Studio wirtualnej maszyny.

Aby skonfigurować maszynę wirtualną najlepiej jest skorzystać z narzędzia AVD menedżer (Android Virtual Device)

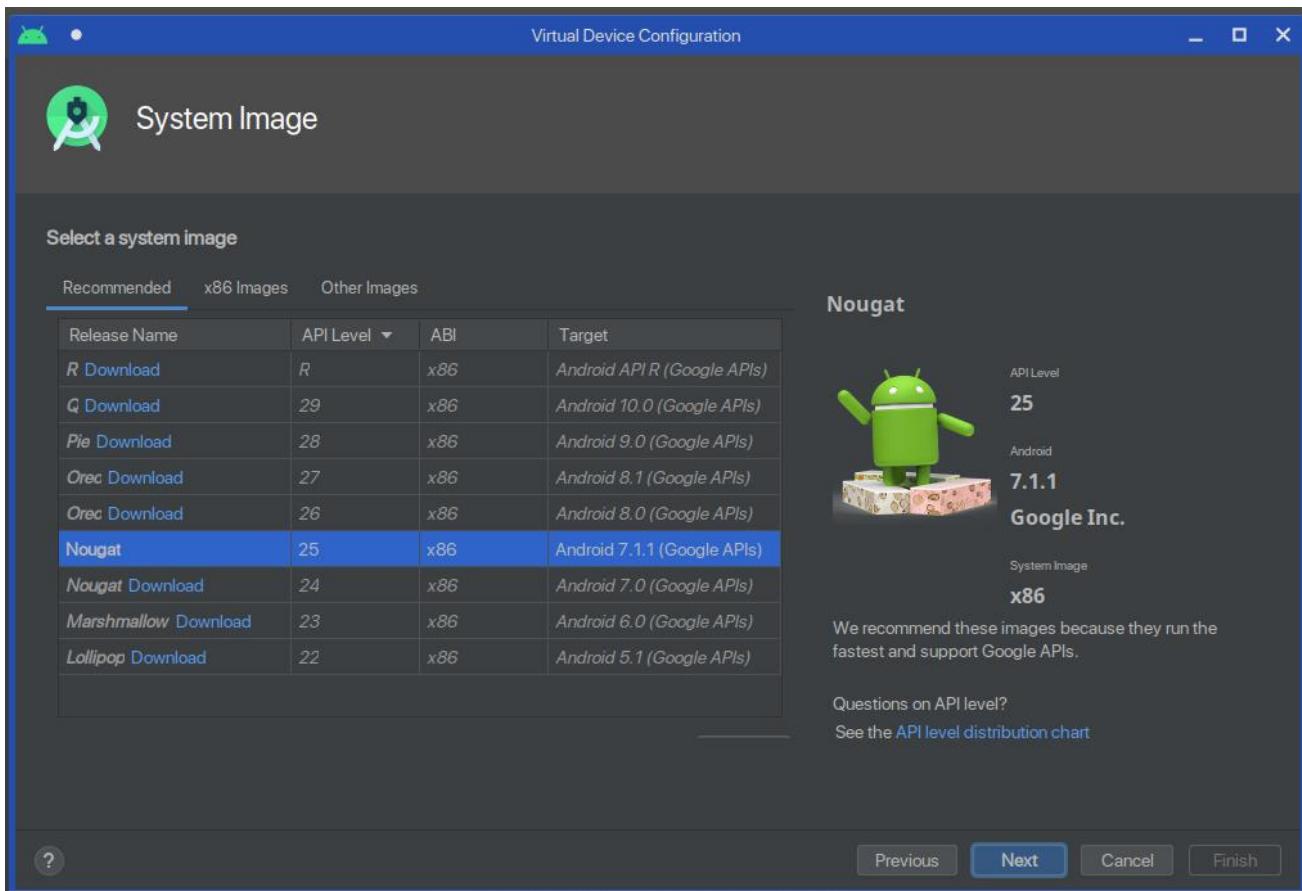


Rysunek 27. Lokalizacja menadżera AVD



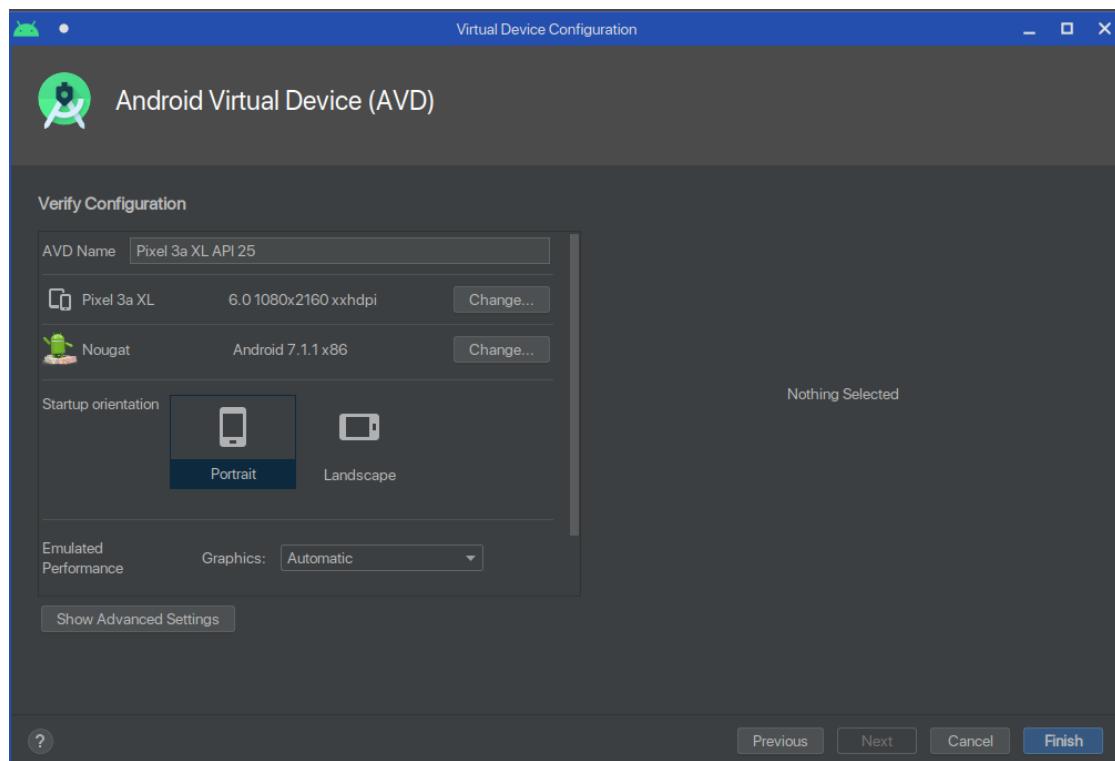
Rysunek 28. AVD menadżer

Do naszych celów możemy wybrać dowolne urządzenie posiadające ekran dotykowy jednak w celach ujednolicenia platform developerskich sugerujemy wykorzystanie urządzenia Pixel 3a XL w trybie pionowym (Portrait).



Rysunek 29. Wybór obrazu systemu wirtualnego urządzenia

Urządzenie może wykorzystywać dowolny system obrazu nowszy niż wersja 24 (Nougat).

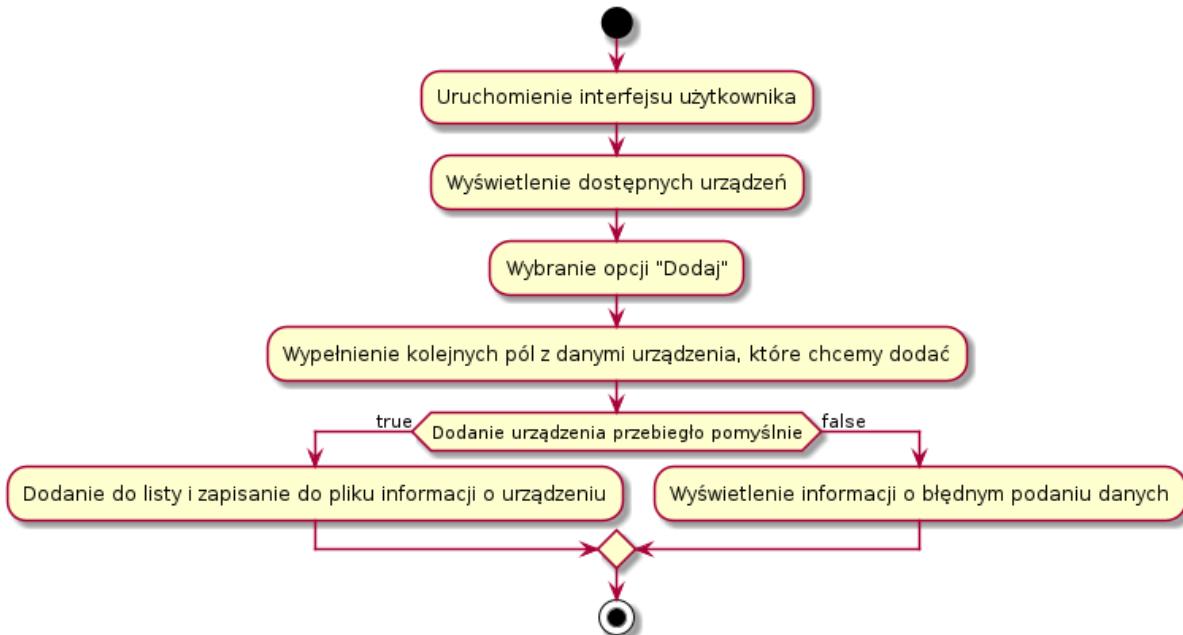


Rysunek 30. Końcowe ustawienia wirtualnego urządzenia.

Po potwierdzeniu poprawności konfiguracji możemy korzystać z aplikacji bez konieczności instalacji jej na prawdziwym sprzęcie.

15.3. Schemat programu

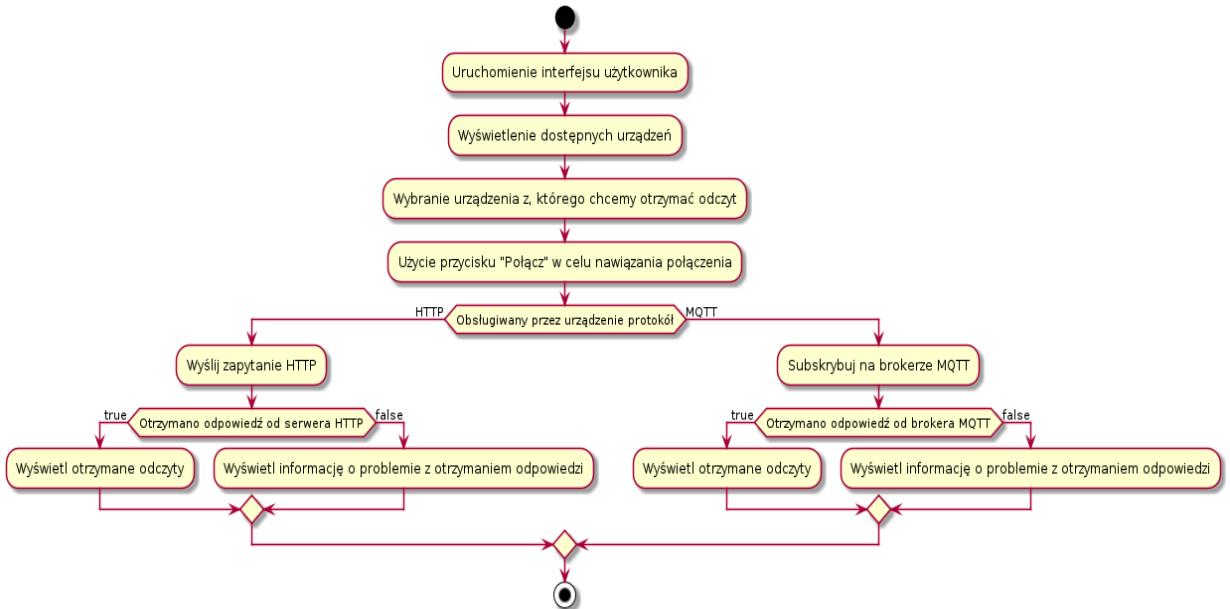
Podstawowe funkcjonalności aplikacji użytkownika na system Android to dodawanie i usuwanie urządzeń IoT oraz wyświetlanie z nich odczytów. Zostały one zamodelowane na następujących diagramach.



Rysunek 31. Dodawanie nowych urządzeń



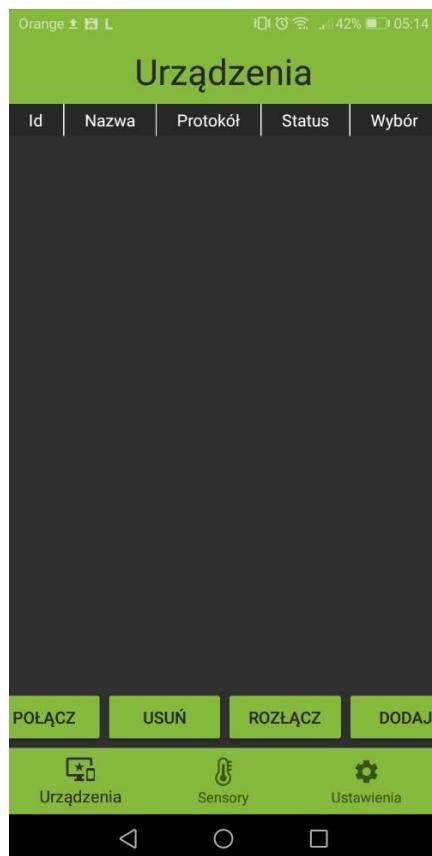
Rysunek 32. Usuwanie urządzeń



Rysunek 33. Odczyt z urządzeń IoT

15.4. Opis oprogramowania z komentarzami i zrzutami ekranu

Zgodnie z powyższymi diagramami zostały utworzone metody pozwalające użytkownikowi na dodawanie, usuwanie i wyświetlanie dostępnych urządzeń oraz wykonywanie z nich odczytów. Utworzony interfejs użytkownika przedstawiony został na poniższym rysunku.



Rysunek 34. Interfejs użytkownika

Przycisk „Dodaj” pozwala na dodawanie nowych urządzeń. Powoduje on uruchomienie nowego okna, które daje możliwość wprowadzenia danych nowego urządzenia.

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    LayoutInflater inflater = getActivity().getLayoutInflater();
    View view = inflater.inflate(R.layout.add_device_dialog, null);
    builder.setView(view)
        .setTitle("Add Device")
        .setPositiveButton("Add",
            new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            MainActivity activity = ((MainActivity) getActivity());

            String full_ip = ip.getText().toString();
            full_ip = full_ip.trim();

            String[] ip_separated = full_ip.split("[.]");
            List<Integer> adres = new ArrayList<>();

            for(int j = 0;j<ip_separated.length;j++){
                adres.add(Integer.parseInt(ip_separated[j]));
            }

            Device device = new Device();

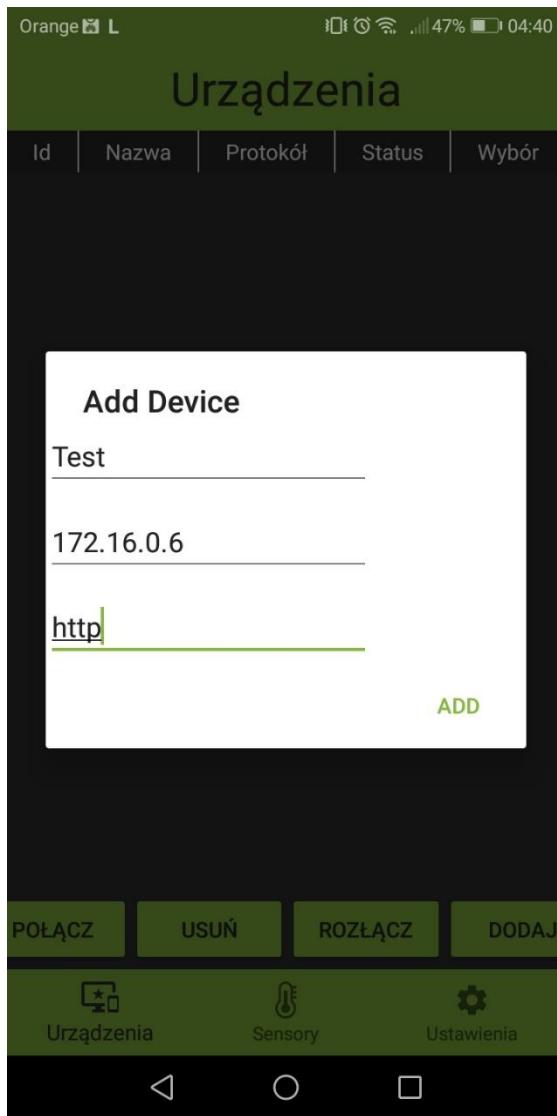
            device.name = name.getText().toString();
            device.protocol = protocol.getText().toString();
            device.address = adres;
            device.id = activity.devices.size();
            activity.devices.add(device);

            activity.display();
        }
    });
}

name = view.findViewById(R.id.editText);
ip = view.findViewById(R.id.editText3);
protocol = view.findViewById(R.id.editText5);

return builder.create();
}

```



Rysunek 35. Dodawanie nowego urządzenia

Przycisk „Usuń” pozwala usunąć z listy urządzeń obecne wybrane urządzenie. Wywołuje on również funkcję display, odpowiedzialną za wyświetlanie dostępnych urządzeń na ekranie.

```
public void deleteDevice(View view) {  
  
    if(selected_device != null){  
        devices.remove(selected_device);  
        display();  
    }  
}
```

Funkcja display, prezentuje się następująco. Tworzy ona nowe obiekty TableRow (kolejne wiersze tabeli wyświetlonej na ekranie) i inicjalizuje je danymi z listy dostępnych urządzeń.

```

@SuppressLint("ResourceType")
public void display()
{
    final TableLayout tl = (TableLayout) findViewById(R.id.device_table);

    if(devices.size() > 0){
        tl.removeAllViewsInLayout();
    }

    for(int i = 0;i<devices.size();i++){
        final TableRow tableRow = new TableRow(MainActivity.this);

        tableRow.setLayoutParams(new TableRow.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.MATCH_PARENT));

        tableRow.setId(1000+i);
        TextView id = new TextView(MainActivity.this);

        String text_id = "" + i;
        id.setText(text_id);

        TextView nazwa = new TextView(MainActivity.this);
        nazwa.setText(devices.get(i).name);

        TextView protokol = new TextView(MainActivity.this);
        protokol.setText(devices.get(i).protocol);

        TextView status = new TextView(MainActivity.this);
        status.setId(1111);

        TextView wybor = new TextView(MainActivity.this);

        tableRow.addView(id);
        tableRow.addView(nazwa);
        tableRow.addView(protokol);
        tableRow.addView(status);
        tableRow.addView(wybor);

        tableRow.setClickable(true);

        tableRow.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                TextView st = (TextView) v.findViewById(1111);

                for (int l = 0; l < devices.size(); l++) {
                    TableRow row = (TableRow) tl.findViewByIndex(1000+l);

                    TextView st2 = (TextView) row.findViewById(1111);

                    st2.setText("");
                }
                st.setText("Wybrano");

                selected_device = devices.get(v.getId()-1000);
            }
        });
    }

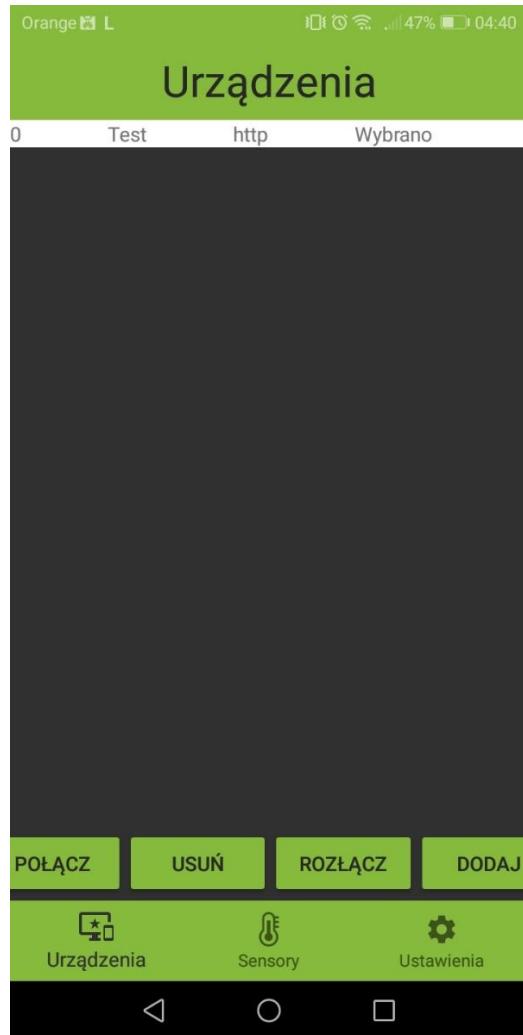
    MainActivity.this.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            tl.addView(tableRow);
        }
    });
}

```

}
});
}

}

Po wybraniu urządzenia, poprzez naciśnięcie na nie, zostaje obok niego wyświetlony komunikat „Wybrano” co umożliwia wykonywanie na nim dalszych operacji („Połącz” lub „Usuń”).



Rysunek 36. Wyświetlanie dodanego urządzenia

Po nawiązaniu połączenia z urządzeniem, możemy przejść do zakładki „Sensory”, gdzie przycisk „Odczyt” pozwala na wyświetlanie kolejnych odczytów z urządzenia.



Rysunek 37. Odczyt danych z urządzenia

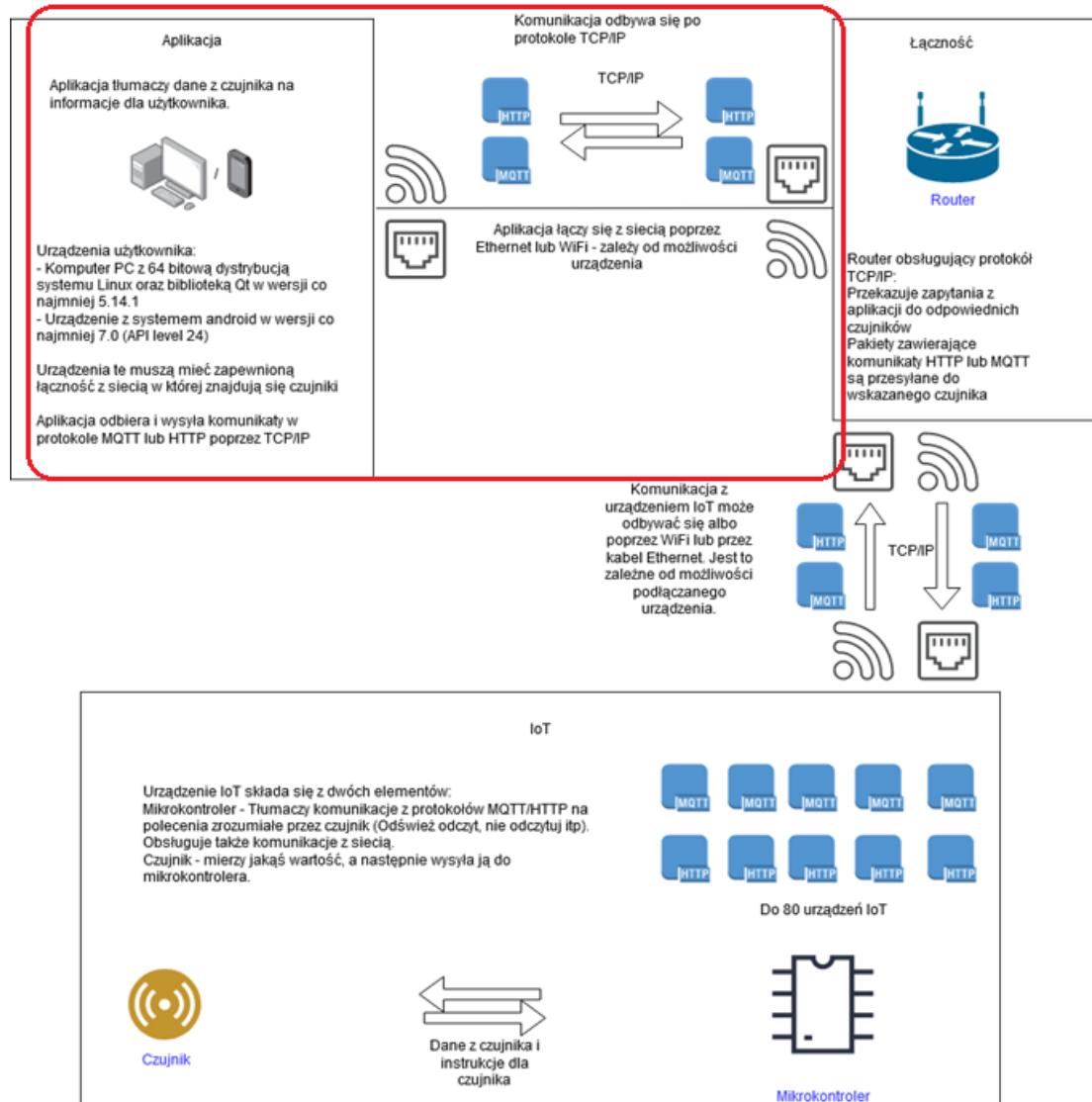
15.5. Uwagi rozwojowe

- Podobnie jak w przypadku aplikacji desktopowej, warto zrezygnować z adresu ip trzymanego jako lista typu int i trzymać go jako zwykły string, dokładamy tylko więcej pracy bo listę intów trzeba konwertować na string przy wysyłaniu zapytań http
- Zadbać żeby sprawdzanie czy urządzenie obsługuje protokół http lub mqtt było case insensitive. Przy dodawaniu nowego urządzenia użytkownik może wpisać Http zamiast http itp.
- Podobny problem przy wpisywaniu adresu ip. Jeśli użytkownik doda spację po ostatnim oktecie to konwersja ze string na int się nie uda. Najpierw trzeba zrobić trim żeby usunąć whitespace a potem zrobić konwersje.

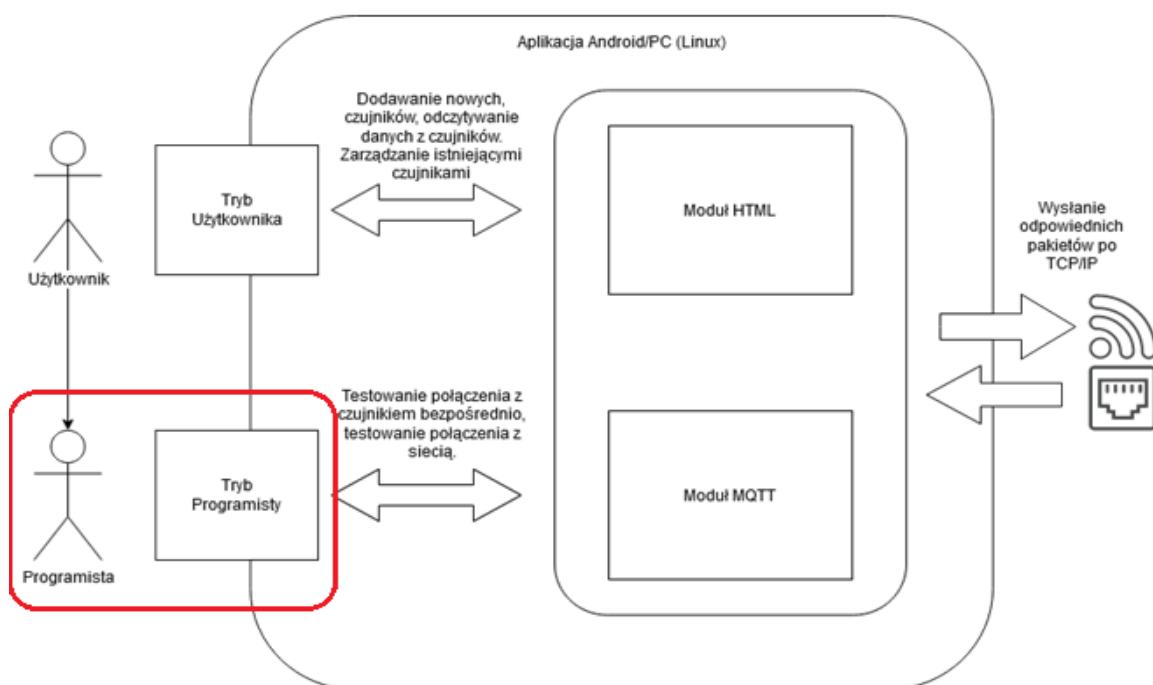
16. Oprogramowanie administratora – WiFi (Tryb administratora w aplikacji na system android)

16.1. Przeznaczenie

Tryb administratora jest wykorzystywany do dokładniejszej analizy działania programu oraz testów połączenia.



Rysunek 38. Aplikacja na system android w schemacie ogólnym

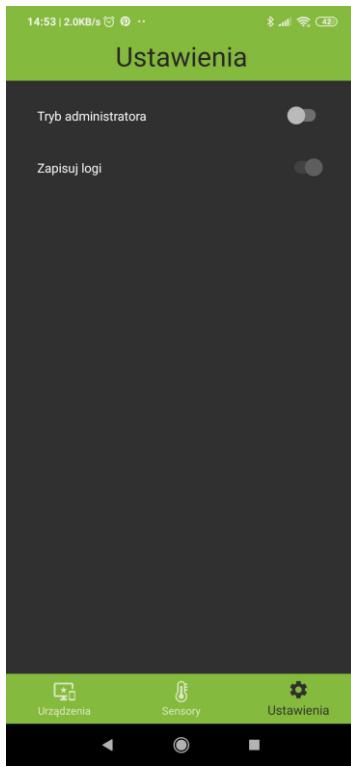


Rysunek 39. Tryb programisty (tryb administratora) aplikacji na system android

16.2. Uruchomienie

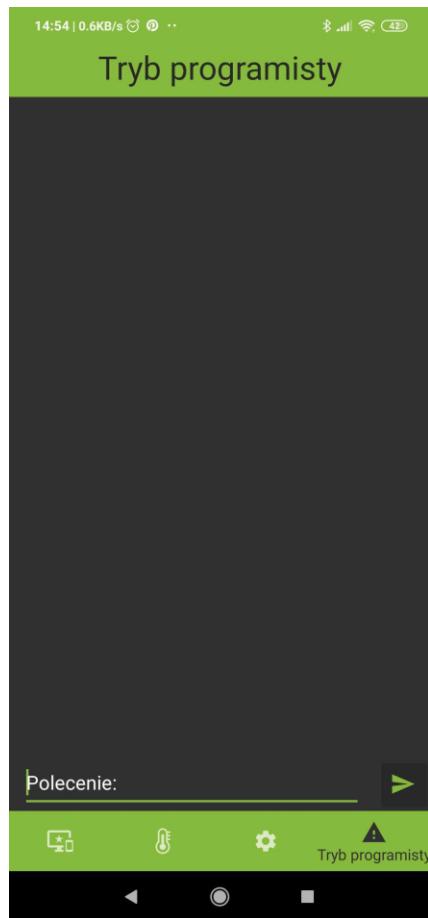
W celu uruchomienia aplikacji proszę patrzyć na punkt [15.2](#)

Dla aplikacji mobilnej aby skorzystać z trybu programisty należy wybrać odpowiednią opcję w ustawieniach:



Rysunek 40. Tryb administratora w ustawieniach aplikacji - Android

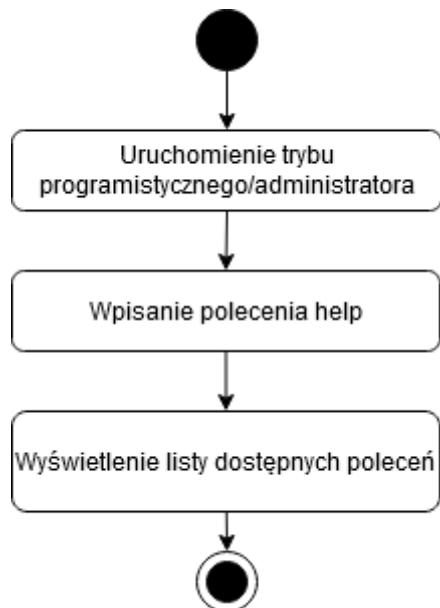
Po odblokowaniu trybu administratora w na dolnym pasku administracyjnym pojawi się nowa opcja oznaczona trójkątem z wykrzyknikiem. Po jej wybraniu znajdziemy terminal administratora.



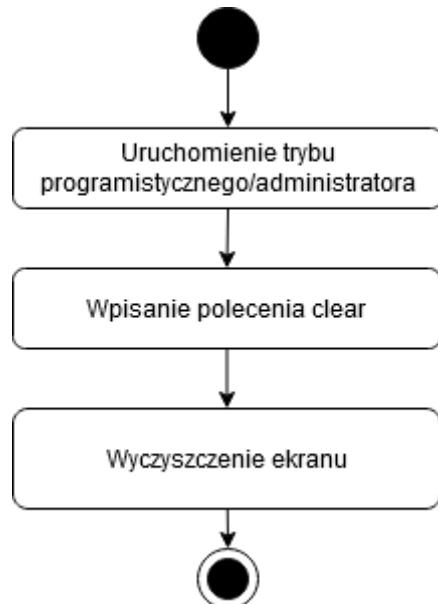
Rysunek 41. Tryb programisty w aplikacji android

16.3. Schemat programu

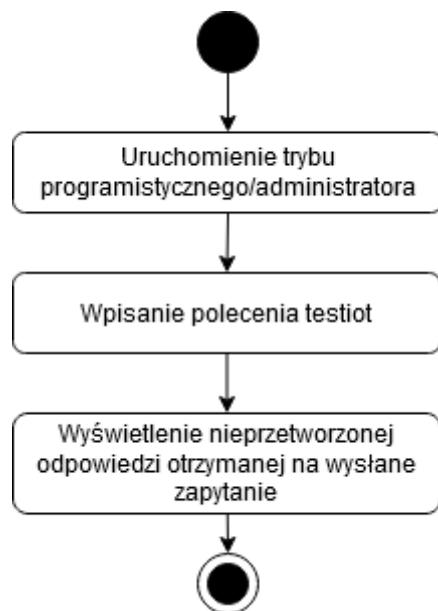
Podstawowe funkcje trybu programisty to testowanie poprawności działania aplikacji:



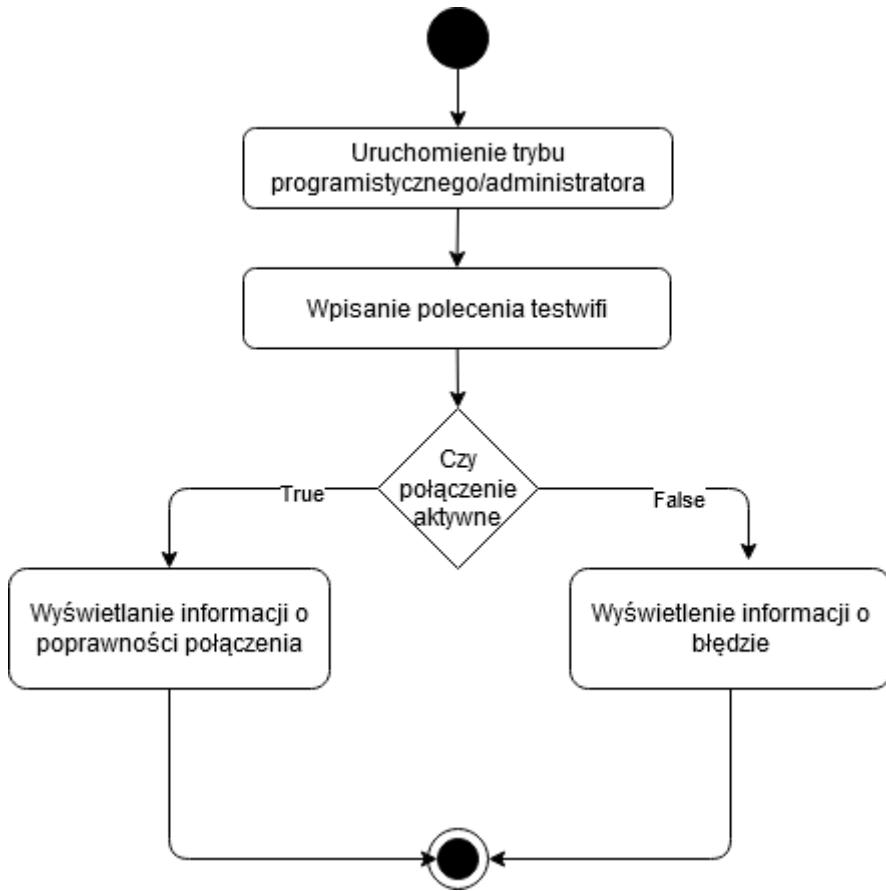
Rysunek 42. Wyświetlenie informacji pomocy



Rysunek 43. Wyczyszczenie historii poleceń



Rysunek 44. Testowanie urządzenia IoT



Rysunek 45. Testowanie połączenia WiFi

16.4. Opis oprogramowania z komentarzami i zrzutami ekranu

Użytkownik ma dostępne następujące polecenia:

A screenshot of a mobile application interface titled "Tryb programisty". The screen shows the following text output:

```

15:38 | 3.0KB/s ⌂ ... 
Tryb programisty

>: help
Dostępne polecenia:
testwifi - Test łączności WiFi
testiot http://0.0.0.0/sensors - Test łączności z IoT
clear - Wyczyszczenie ekranu

Polecenie: >

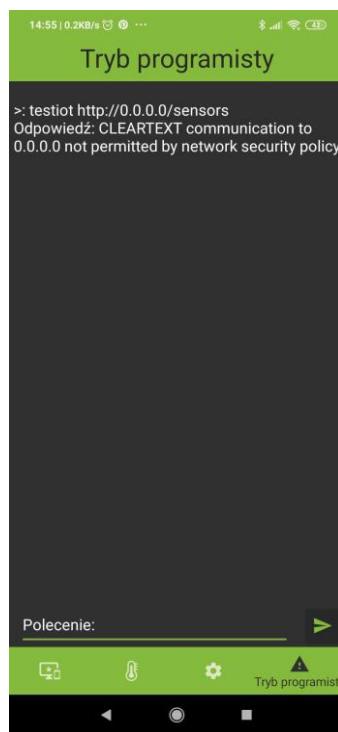
```

The interface has a green header bar with the title "Tryb programisty". Below the header is a black text area displaying the command history and available commands. At the bottom is a green footer bar with icons for battery, signal, and settings, and the text "Tryb programisty".

Rysunek 46. Polecenie help

help – wyświetla wszystkie dostępne w aplikacji polecenie wraz z krótką instrukcją ich użycia.

```
if(text.equals("help"))
{
    outText.append("\nDostępne polecenia:");
    outText.append("\ntestwifi - Test łączności WiFi \ntestiot http://0.0.0.0/sensors - Test łączności z IoT\nclear - Wyczysć ekran");
}
```



Rysunek 47. Polecenie test IoT

testiot – testuje połączenie bezpośrednio z iot, a jako argument przyjmuje adres urządzenia.

```
String[] separated = text.split(" ");

if(separated[0].equals("testiot"))
{
    if(separated.length > 1)
    {
        String url = separated[1];

        OkHttpClient okHttpClient = new OkHttpClient();

        Request request = new Request.Builder().url(url).build();
        okHttpClient.newCall(request).enqueue(new Callback() {@Override
public void onFailure(@NotNull Call call, @NotNull IOException e) {
    final String myResponse = e.getMessage().toString();

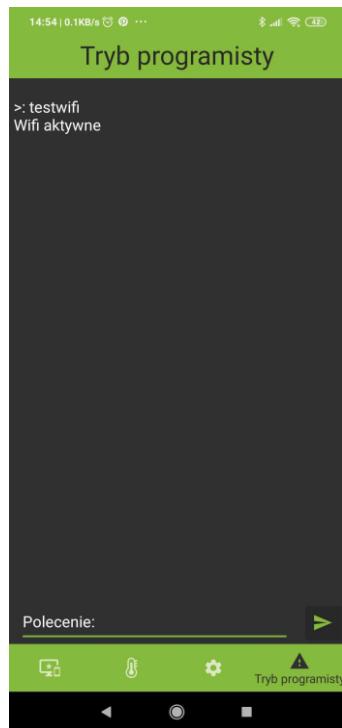
    MainActivity.this.runOnUiThread(new Runnable() { @Override
```

```
public void run() {
    outText.append("\nOdpowiedź: ");
    outText.append(myResponse);
}
});

}

@Override
public void onResponse(@NotNull Call call, @NotNull Response response) throws IOException {
    if(response.isSuccessful())
    {
        final String myResponse = response.body().string();

        MainActivity.this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                outText.append("\nOdpowiedź: ");
                outText.append(myResponse);
            }
        });
    }
    else
    {
        MainActivity.this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                outText.append("\nProblem z połączeniem z urządzeniem IoT");
            }
        });
    }
}
}
```



Rysunek 48. Test dzialania sieci

testwifi – testuje połaczenie z siecią.

```
if(text.equals("testwifi"))
{
    String url = "https://postman-echo.com/get?foo1=bar1&foo2=bar2";
    OkHttpClient okHttpClient = new OkHttpClient();

    Request request = new Request.Builder()
        .url(url)
        .build();

    okHttpClient.newCall(request).enqueue(new Callback()
    {
        @Override
        public void onFailure(@NotNull Call call, @NotNull IOException e)
        {
            final String myResponse = e.getMessage().toString();

            MainActivity.this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    outText.append("\nProblem z połączeniem WiFi");
                }
            });
        }
    });
}
```

```

    }

    @Override
    public void onResponse(@NotNull Call call, @NotNull Response response) throws IOException {
        if(response.isSuccessful())
        {
            final String myResponse = response.body().string();

            MainActivity.this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    outText.append("\nWifi aktywne");
                }
            });
        }
        else
        {
            MainActivity.this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    outText.append("\nProblem z połączeniem WiFi");
                }
            });
        }
    });
}

```

clear – czyści historię wpisanych poleceń.

```
if(text.equals("clear")) outText.setText("");
```

16.5. Uwagi rozwojowe

- Na tym etapie wszystkie funkcje zostały zaimplementowane w tej części programu.

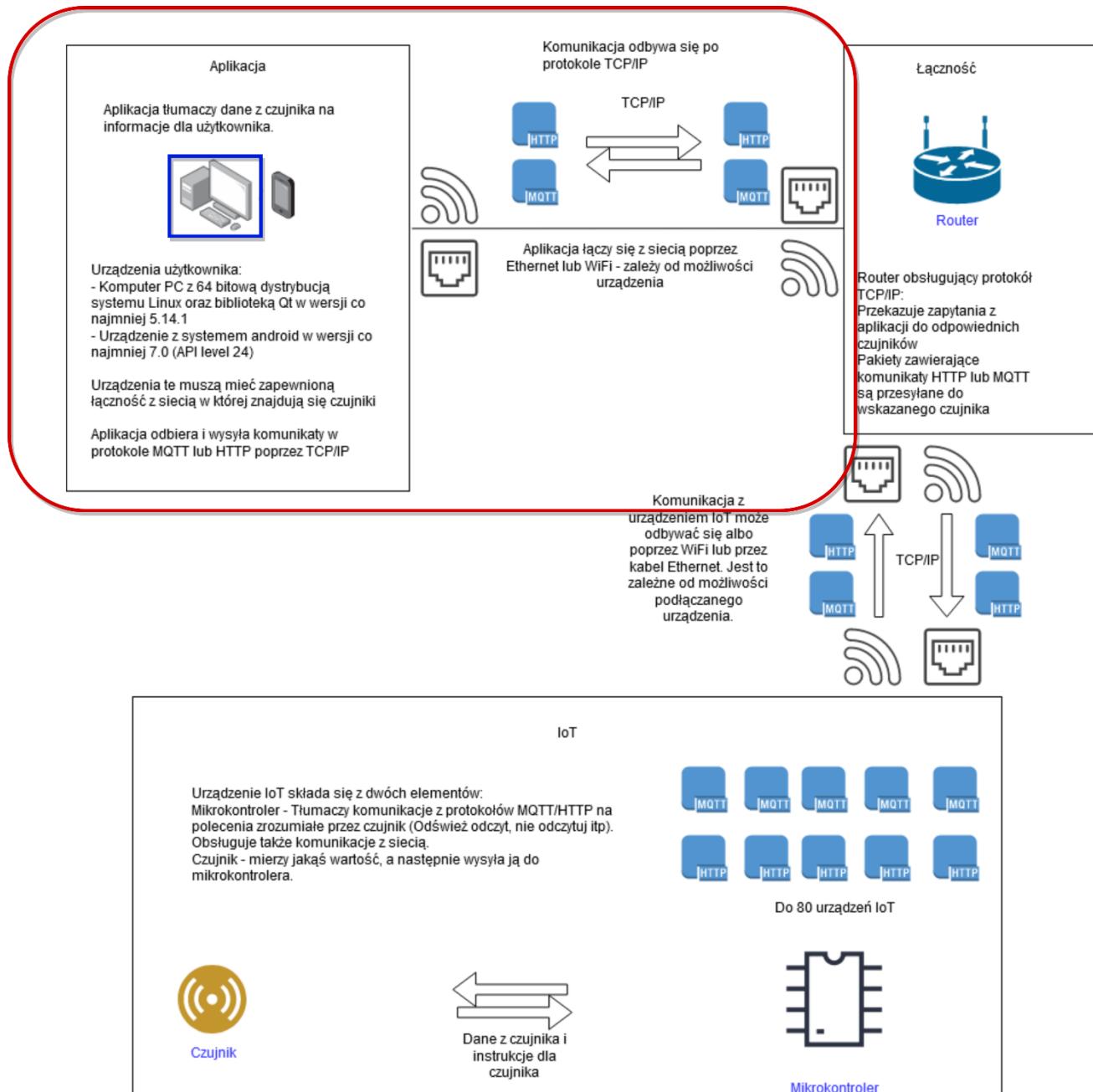
17. Oprogramowanie użytkownika Router WiFi (Tryb użytkownika w aplikacji na system Linux)

Wykonanie: Katarzyna Czajkowska/Mateusz Gurski - aplikacja i Adam Krizar – wirtualizacja Linuxa
Sprawdzenie: Mateusz Gurski

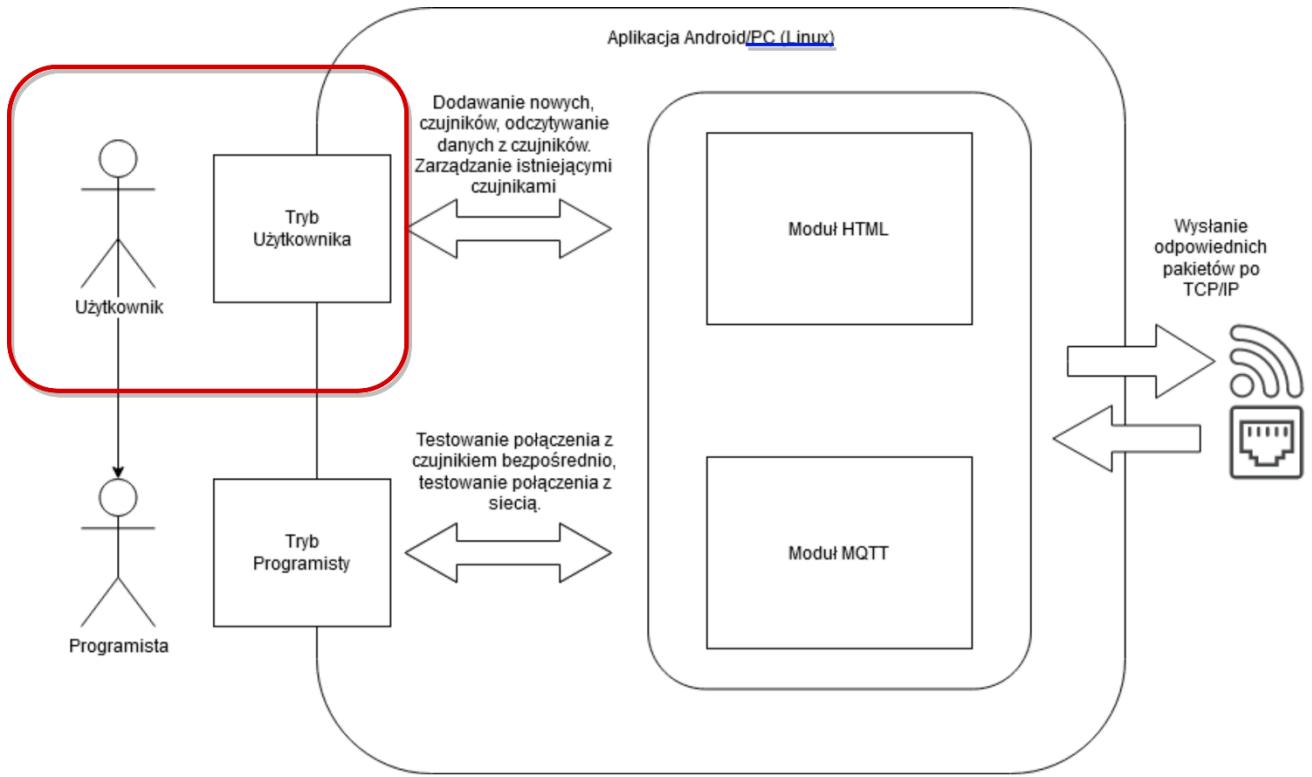
17.1. Przeznaczenie

Rola trybu użytkownika w aplikacji desktopowej jest taka jak w przypadku trybu użytkownika w aplikacji mobilnej czyli - pozwala na dodawanie nowych czujników, zarządzanie istniejącymi i wyświetlanie odczytów

z czujników wykorzystując protokół komunikacji HTTP lub MQTT. Rola tego oprogramowania w schemacie ogólnym przedstawiona została poniżej.



Rysunek 49. Rola aplikacji na system Linux w schemacie ogólnym



Rysunek 50. Tryb użytkownika aplikacji na system Linux

17.2. Uruchomienie

Obecna instrukcja zakłada wykorzystanie systemu operacyjnego Manjaro Linux z środowiskiem KDE. Jest on dostępny do pobrania tutaj: <https://manjaro.org/downloads/official/kde/>.

Zaleca się pobranie pełnej wersji systemu ponieważ wymaga ona najmniej przygotowań ze strony użytkownika.

Do testowania aplikacji potrzebny jest także plik IoTMan.AppImage, który zawiera odpowiednio spakowaną aplikację wraz ze wszystkimi wymaganymi bibliotekami.

17.2.1. Uruchomienie Linuxa w Oracle Virtual Box

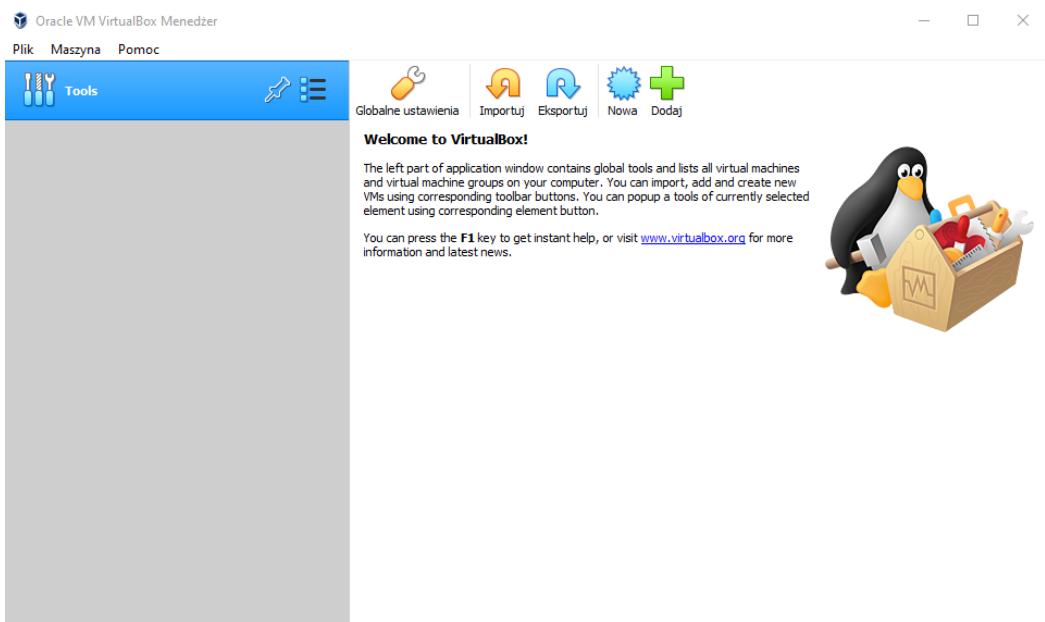
UWAGA: Aby wykorzystać tę opcję konieczne jest włączenie wirtualizacji na płycie głównej. Można sprawdzić czy ta opcja jest uruchomiona w menadżerze zadań w Windowsie 10 w zakładce Wydajność -> CPU.

Procesory logiczne:	12
Wirtualizacja:	Włączone
Dostęp podirectny poziomu 1:	576 KB

Rysunek 51. Status wirtualizacji

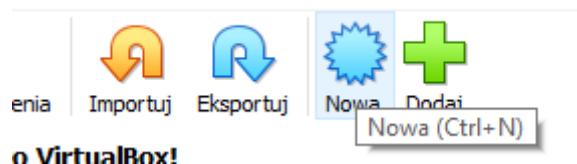
Do wirtualizacji systemu Linux potrzebne oprogramowanie Oracle Virtual Box dostępne pod adresem: <https://www.virtualbox.org/wiki/Downloads>. Zalecamy pobranie wersji 6.1.6 dla Windows hosts.

Aby zainstalować wymagane oprogramowanie wystarczy postępować zgodnie z instrukcjami instalatora.



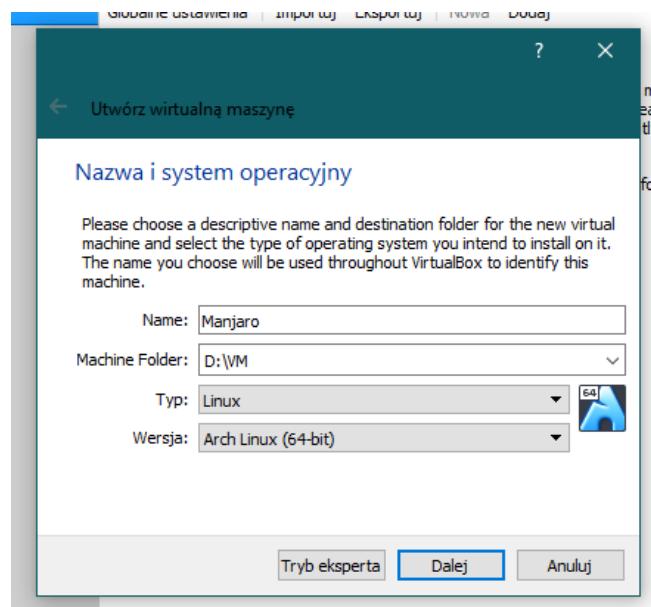
Rysunek 52. Interfejs programu VirtualBox

W celu utworzenia nowej maszyny wirtualnej naciskamy przycisk nowa lub skrót ctrl + n.



Rysunek 53. Utworzenie nowej maszyny wirtualnej

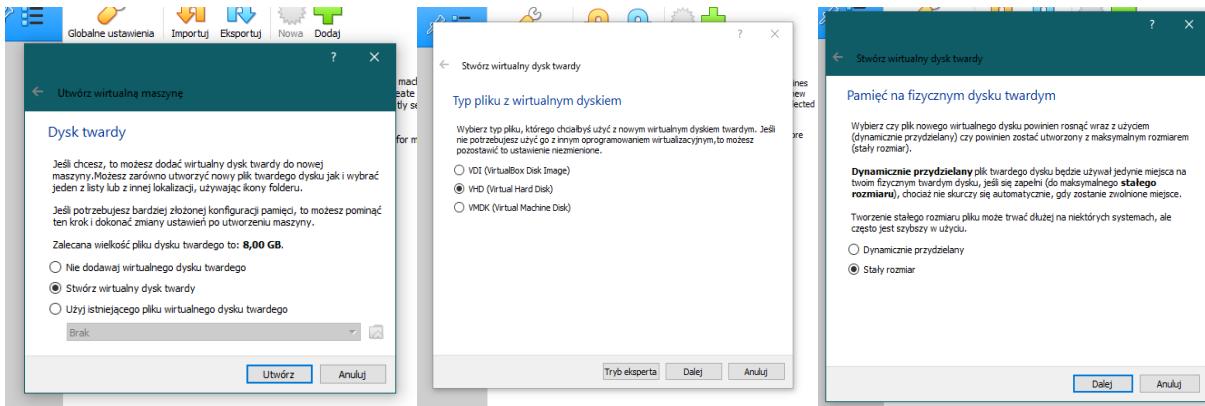
W oknie, które w ten sposób otworzyliśmy wybieramy następujące rzeczy:



Rysunek 54. Tworzenie maszyny wirtualnej

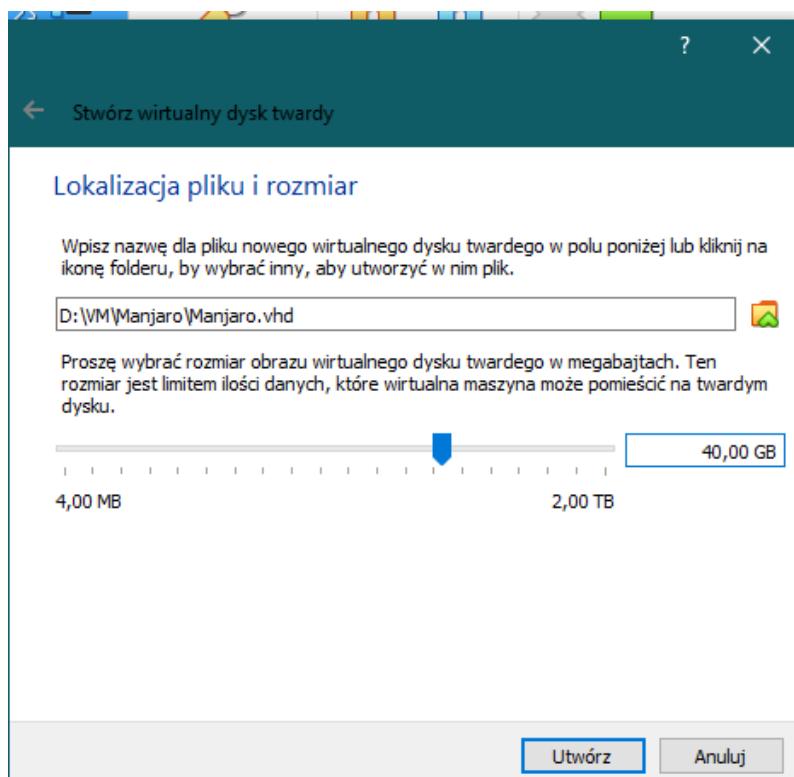
Nazwę i folder maszyny według własnego uznania. Typ Linux i wersje na Arch Linux (64-bit).

Następnie wybieramy ilość pamięci RAM, którą chcemy przydzielić maszynie. Optymalnie zalecamy użycie 4GB RAM-u, ale w przypadku braku dostępności takiej ilości minimalnie wystarczą 2 lub 1 GB.



Rysunek 55. Tworzenie dysku twardego

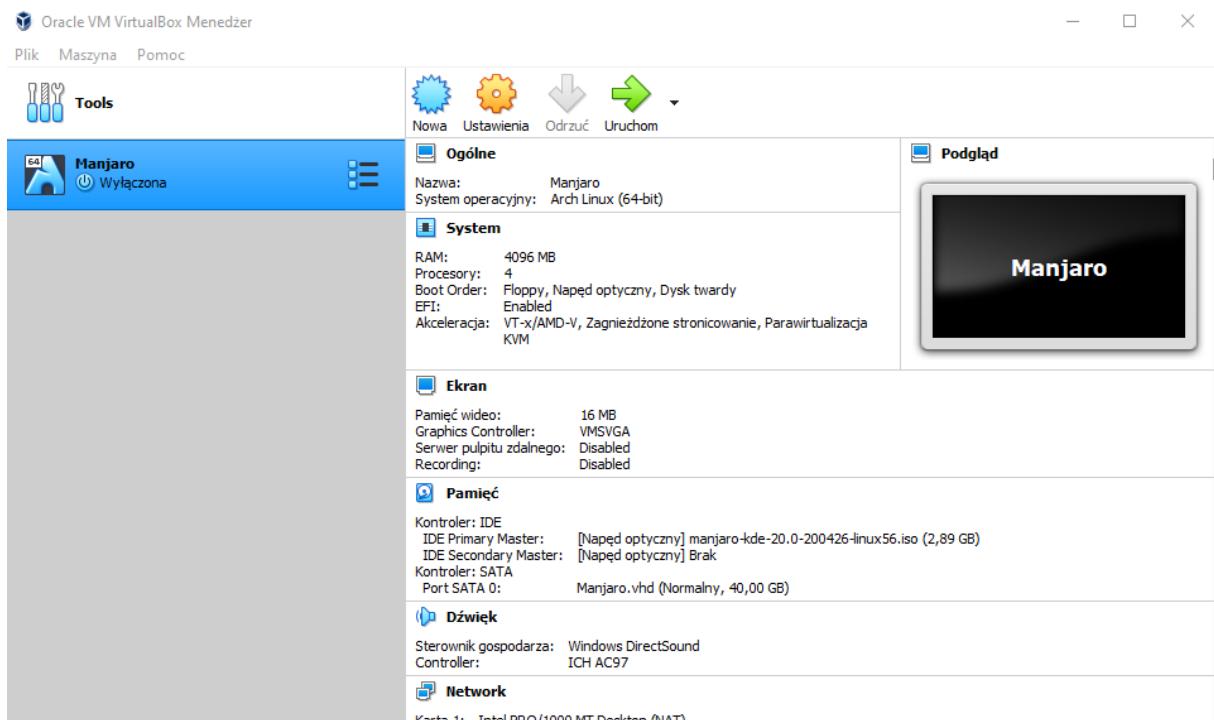
W kolejnym kroku musimy przydzielić dysk dla naszej maszyny testowej. Na pierwszych dwóch ekranach wybieramy domyślne opcje, natomiast na trzecim zmieniamy z dynamicznie przydzielany na stały rozmiar.



Rysunek 56. Ustawienie rozmiaru dysku twardego.

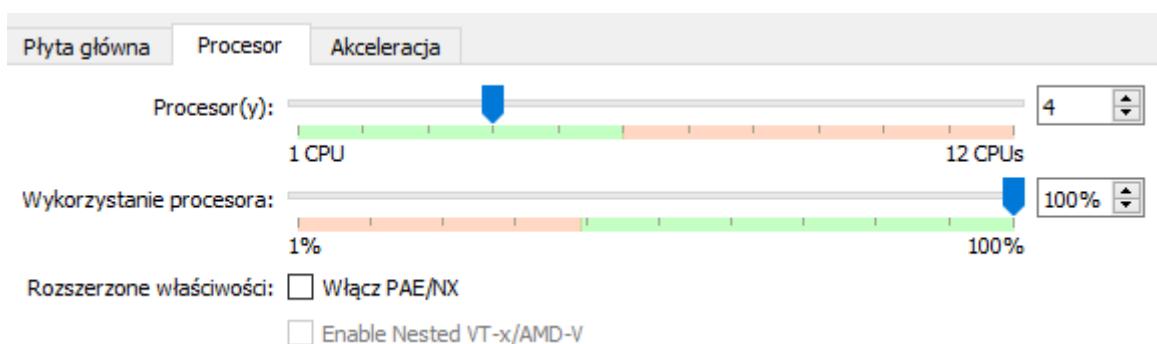
Zalecanym rozmiarem dysku jest przedział od 15 do 20 GB. Po kliknięciu utwórz musimy uzbroić się w cierpliwość gdy komputer alokuje wybraną przez nas przestrzeń dyskową.

Po zakończeniu tej operacji maszyna wirtualna zostanie wstępnie utworzona.



Rysunek 57. Utworzona maszyna wirtualna.

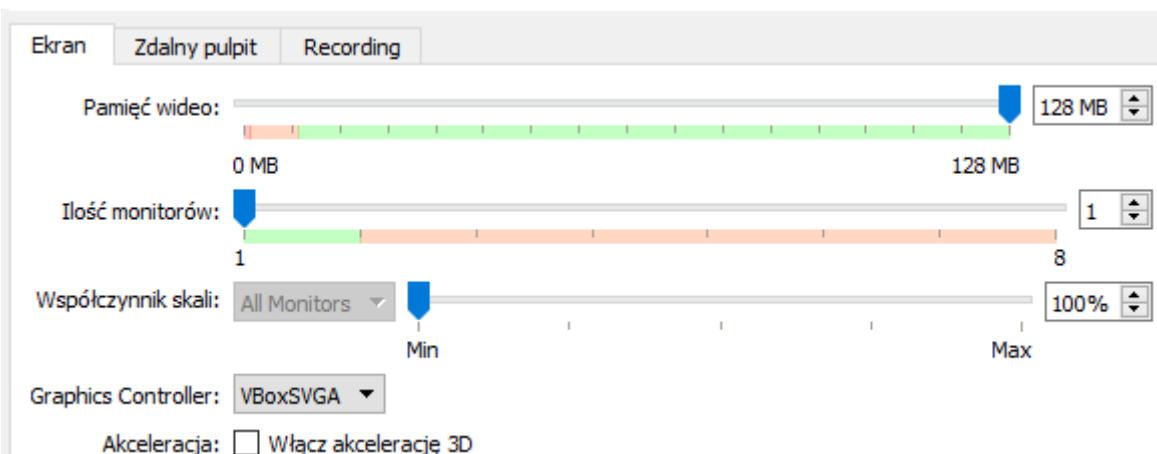
Nie uruchamiamy jeszcze maszyny. Wybieramy ustawienia -> System-> Procesor.



Rysunek 58. Ustawienia procesora

Zalecamy przypisane co najmniej dwóch rdzeni do maszyny wirtualnej.

Kolejno przechodzimy do zakładki ekran. Zmieniamy ilość pamięci wideo na maximum 128 MB, graphics Controller ustawiamy na VBoxSVGA i odznaczamy włącz akcelerację 3D.



Rysunek 59. Ustawienia ekranu.

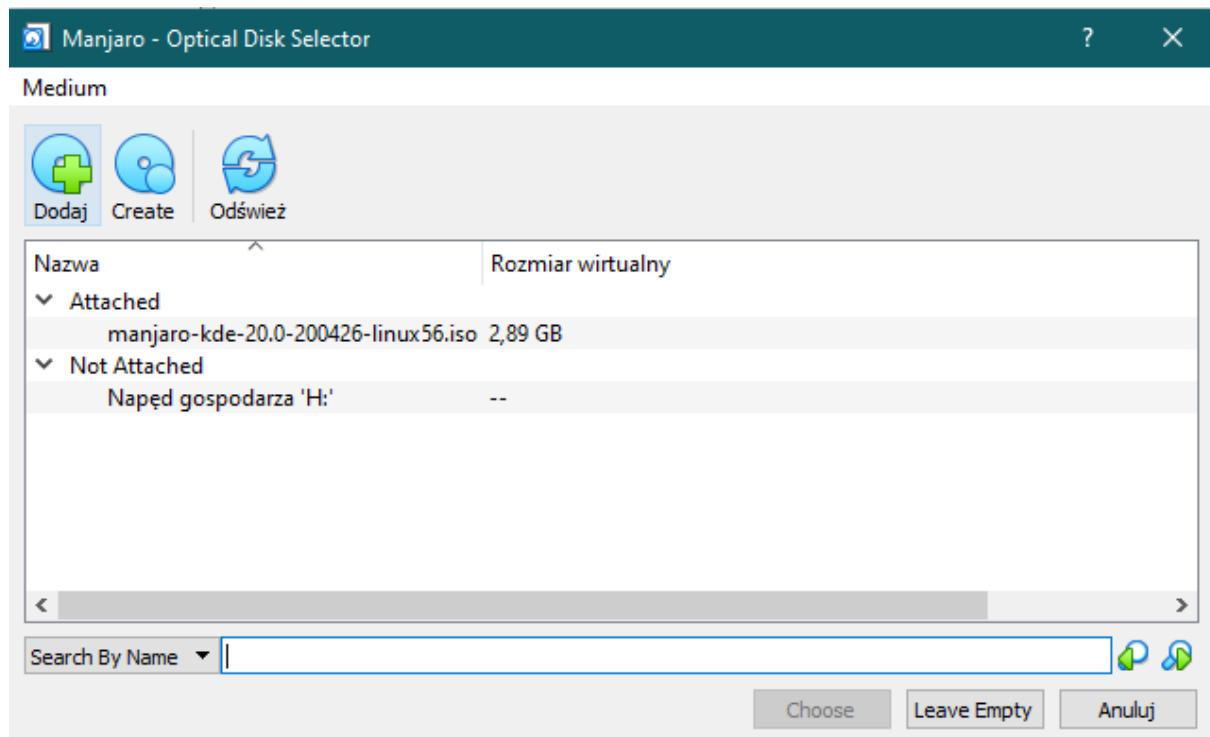
Następnie przechodzimy w zakładkę pamięć.

Wybieramy tam przycisk dodaj napęd optyczny w zakładce Kontroler: IDE.



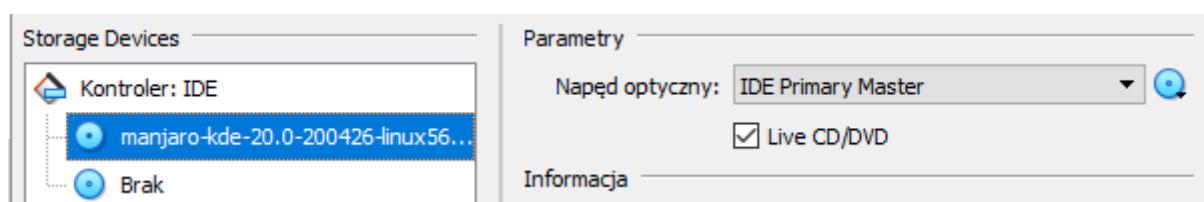
Rysunek 60. Utworzenie wirtualnego napędu optycznego.

W oknie które się na otworzyło wybieramy przycisk dodaj.



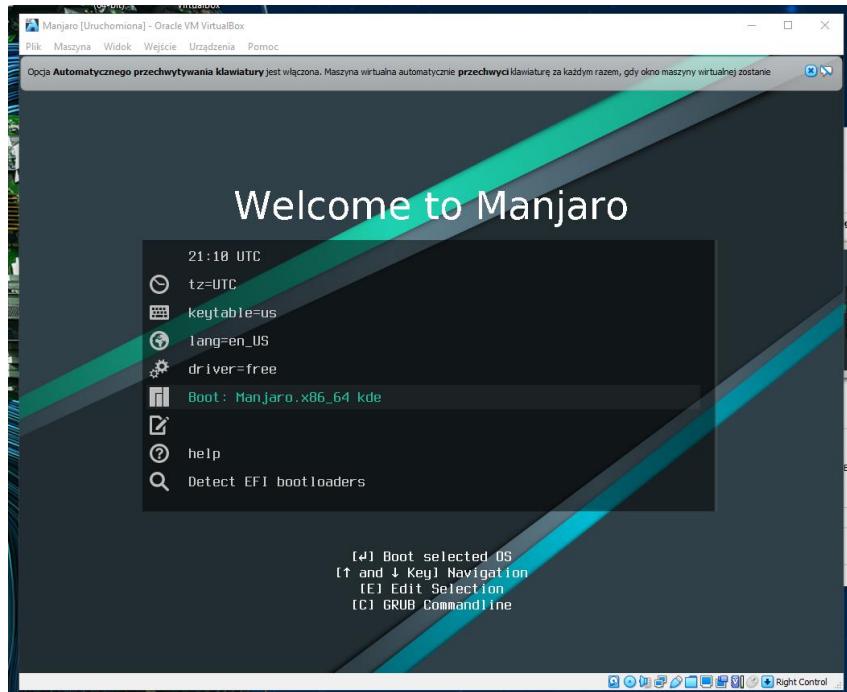
Rysunek 61. Wybór obrazu systemu.

W oknie systemowym które się otworzyło nawigujemy do lokacji gdzie zapisaliśmy obraz systemu i go wybieramy.



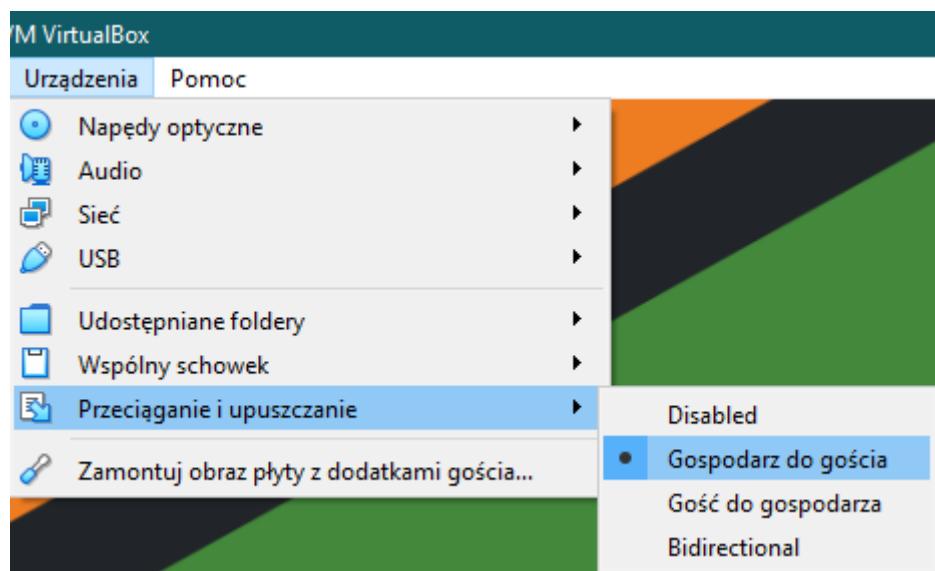
Rysunek 62. Ustawienie opcji Live CD/DVD

Po utworzeniu wirtualnego napędu konieczne jest żebyśmy w parametrach zaznaczyli opcję Live CD/DVD.



Rysunek 63. Rysunek 15 Uruchomiona maszyna wirtualna

Klikamy na zakładkę urządzenia. Wybieramy opcje przeciąganie i upuszczanie i zaznaczmy gospodarz do goszcza. Umożliwi nam to udostępnienie plików dla systemu Linux.



Rysunek 64. Udostępnianie plików

17.2.2. Uruchomienie systemu Linux na komputerze

System Linux oferuje możliwość przetestowania oprogramowania bez instalacji systemu na dysku twardym. Aby przygotować dysk USB z którego moglibyśmy przetestować aplikacje konieczne są dwie rzeczy:

- Pendrive minimum 4 GB
- Aplikacja Rufus dostępna pod adresem <https://rufus.ie/>

W aplikacji Rufus wybieramy przycisk WYBIERZ aby wybrać obraz systemu:



Rysunek 65. Wybór obrazu systemu

Następnie naciskamy start -> operacja ta automatycznie wybierze dysk USB i wymarzę jego zawartość.

Po zakończeniu tej operacji upewniamy się że nasz komputer pozwala na start z dysku USB, a następnie wybieramy nasz pendrive jako dysk startowy w BIOS.

17.2.3. Uruchomienie aplikacji



Rysunek 66. Uruchomiony system Linux

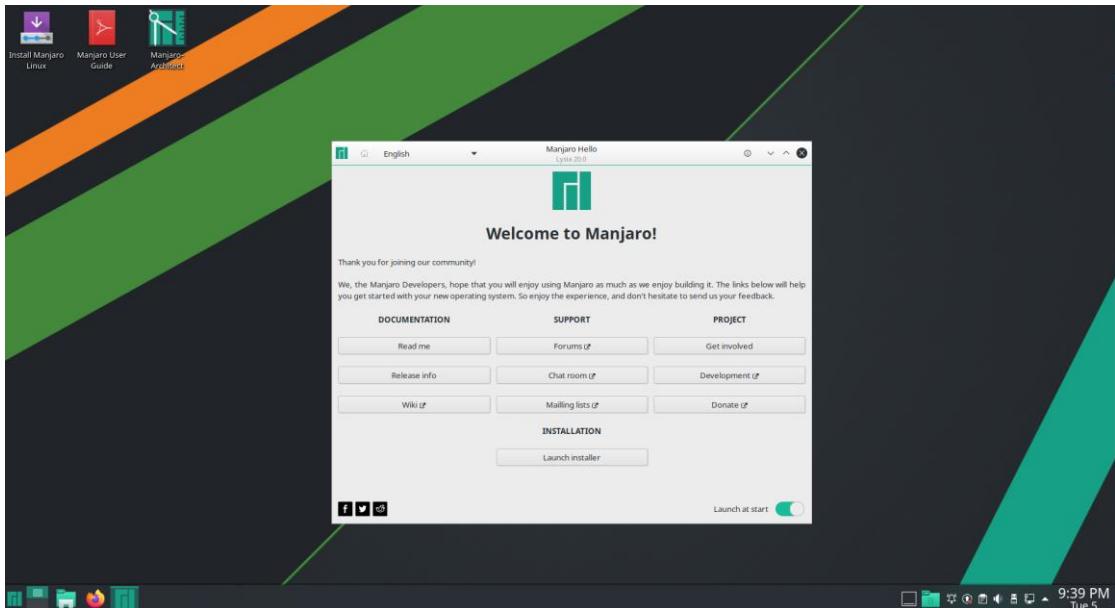
Po uruchomieniu systemu Manjaro konieczne są następujące kroki.

W zakładce keytable=us zmieniamy układ klaviatury na keytable=pl.

Jeżeli uruchamiamy system na prawdziwym sprzęcie z kartą graficzną nvidii konieczna będzie zmiana z driver=free na driver=nonfree.

Po zmienieniu tych rzeczy wybieramy opcje Boot: Manjaro.x86_64_kde.

Naszym oczom ukaże się interfejs systemu operacyjnego.



Rysunek 67. System Manjaro Linux

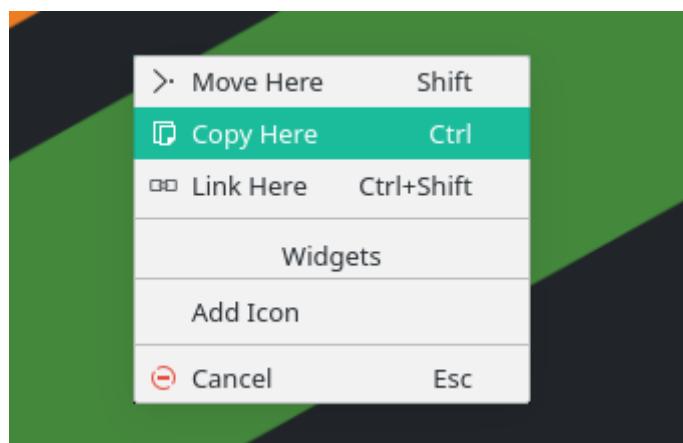
Zamykamy okno Manjaro Hello.

Jeżeli jesteśmy na maszynie wirtualnej przeciągamy na pulpit systemu Linux plik .AppImage

Jeżeli korzystamy z prawdziwego komputera nawigujemy do dysku na którym mamy zapisany plik .AppImage, a następnie kopujemy go na pulpit. Możemy to zrobić z poziomu aplikacji Dolphin, która uruchamia się naciskając ikonkę folderu na pasku zadań.



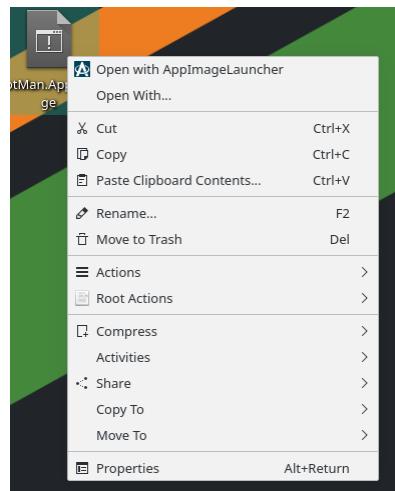
Rysunek 68. ikona programu Dolphin



Rysunek 69. Kopiowanie pliku aplikacji

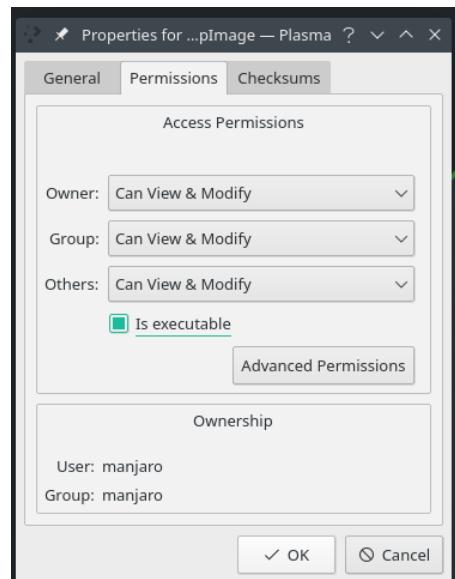
Wybieramy opcję copy here.

Następnie przy pomocy prawego przycisku myszy otwieramy menu Properties.



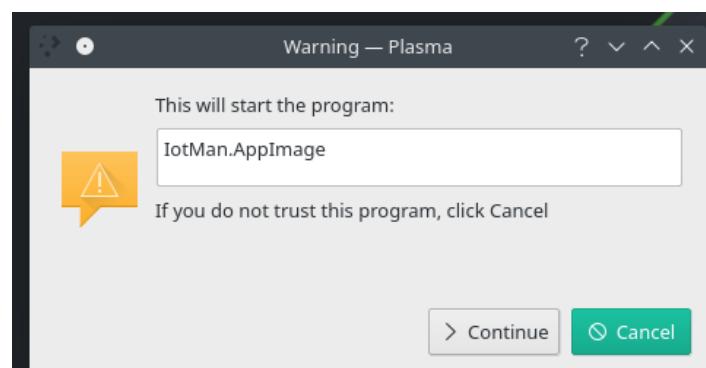
Rysunek 70. Menu Properties

W zakładce Permissions wybieramy Is executable i klikamy ok.



Rysunek 71. Nadanie praw aplikacji

Następnie podwójnie klikamy na aplikację i wybieramy continue na ostrzeżeniu, które się pojawiło.



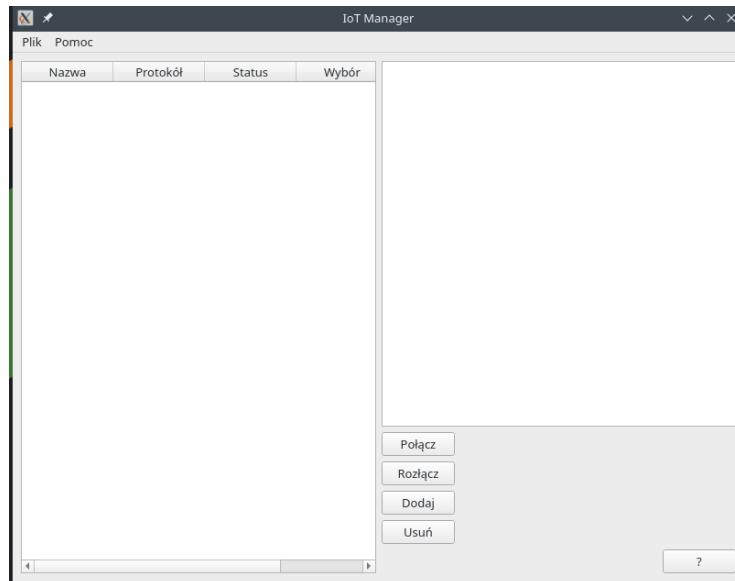
Rysunek 72. Uruchomianie aplikacji

To samo robimy w następnym oknie, klikamy ok.



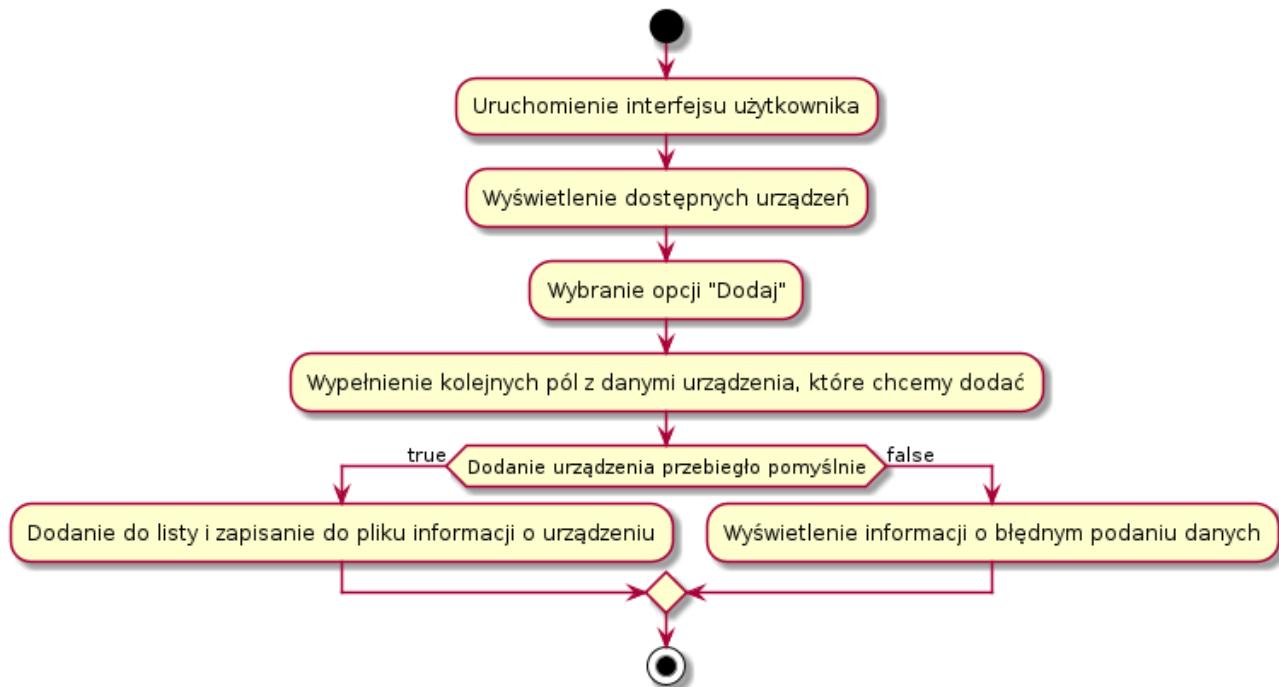
Rysunek 73. Uruchomianie aplikacji

Po tych operacjach otworzy się okno aplikacji. Możemy z niej normalnie korzystać.

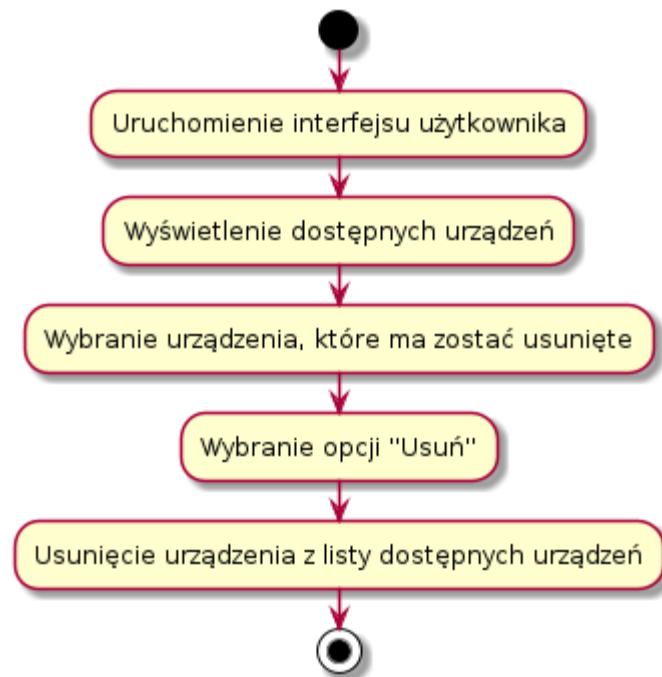


Rysunek 74. Uruchomiona aplikacja

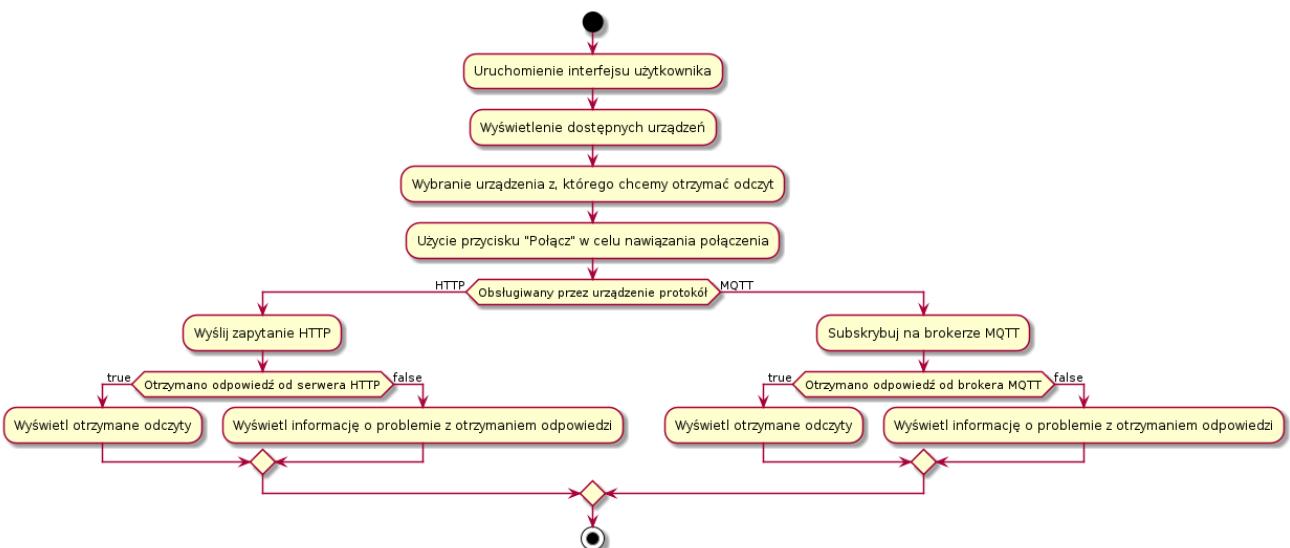
17.3. Schemat programu



Rysunek 75. Dodawanie urządzeń.



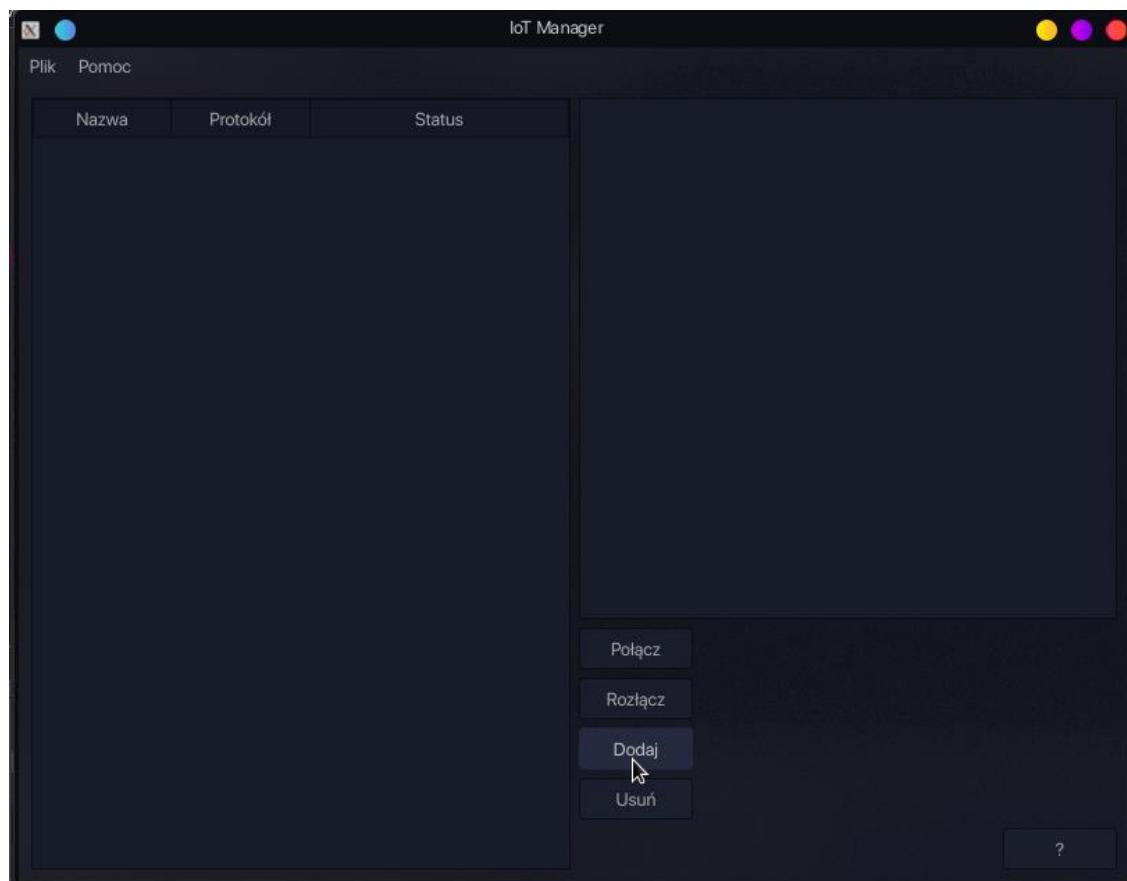
Rysunek 76. Usuwanie urządzeń.



Rysunek 77. Wyświetlanie odczytów

17.4. Opis oprogramowania wraz z komentarzami i zrzutami ekranów

17.4.1. Podstawowe funkcje aplikacji desktopowej



Rysunek 78. Główne okno aplikacji desktopowej

Wyświetlanie tabeli urządzeń:

```
void UserInterface::display()
{
    if (devices.size() <= 0) return;
    else
    {
        QComboBox *pBox;
        QLabel *img;
        QPixmap pmap("x.png");
        pmap = pmap.scaled(20, 20, Qt::IgnoreAspectRatio, Qt::SmoothTransformation);
        ui->iot_table->setRowCount(devices.size());
        for (int i = 0; i < devices.size(); i++)
        {
            pBox = new QComboBox();
            pBox->addItem("HTTP");
            pBox->addItem("MQTT");
            img = new QLabel;
            img->setPixmap(pmap);
            img->setAlignment(Qt::AlignCenter);
            if (devices[i]->getProtocol() == 0) pBox->setcurrentIndex(1);
            ui->iot_table->setItem
                (i, 0, new QTableWidgetItem(devices[i]->getName()));
            ui->iot_table->setCellWidget(i, 1, pBox);
            ui->iot_table->setCellWidget(i, 2, img);
        }
    }
}
```

Dokładny wygląd okna zależy od dystrybucji Linuxa, na której program jest uruchomiony. Zrzut ekranu przedstawia program uruchomiony na Manjaro.

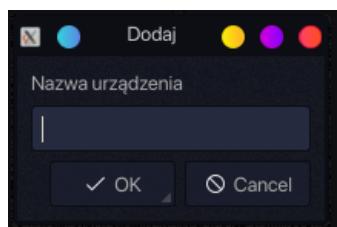
Jeżeli byłoby podłączone urządzenie, informacje na jego temat pojawiłyby się po lewej stronie okna, która służy za tabelę. Uruchomienie programu z przykładowym urządzeniem IoT pokazane jest w rozdziale 10.2, gdzie również pokazane jest działanie przycisków „**Połącz**” i „**Rozłącz**”.

Wyjaśnienie zawartości tabeli:

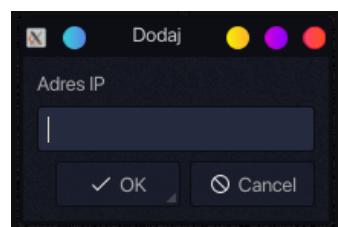
- **Nazwa** – pokazuje nazwę przypisaną urządzeniu, które mamy wprowadzone. Służy do celu identyfikacji urządzenia przez użytkownika;
- **Protokół** – daje możliwość wyboru protokołu, przez jaki chcemy się połączyć z danym urządzeniem;
- **Status** – wyświetla aktualny stan urządzenia - ✓ oznacza, że urządzenie jest połączone, X oznacza, że urządzenie jest rozłączone;

Odczyt z urządzeń pojawi się po prawej stronie, w polu nad przyciskiem „Połącz”.

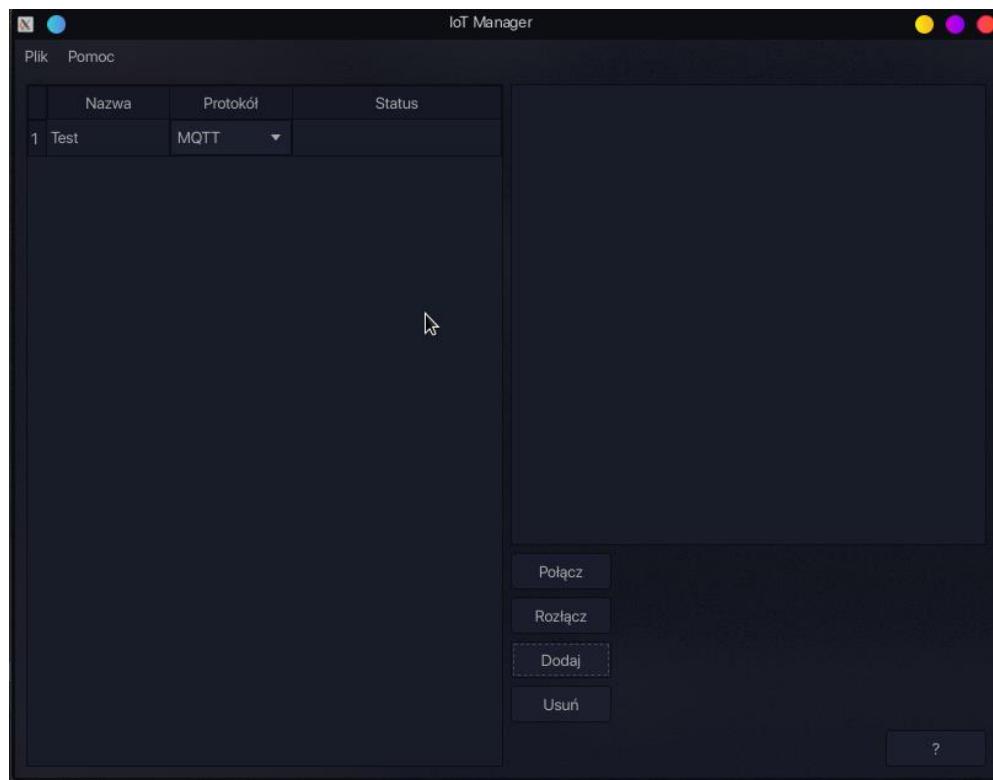
Przycisk „**Dodaj**” wyświetla okna dialogowe pozwalające na wprowadzenie nazwy oraz adresu IP dodawanego urządzenia:



Rysunek 79. Dodawanie nazwy urządzenia



Rysunek 80. Dodawanie IP urządzenia.



Rysunek 81 .Okno aplikacji po dodaniu nowego urządzenia

```
void UserInterface::on_addButton_clicked()
{
    if (helpmode)
    {
        QMessageBox info;
        info.setWindowTitle("Pomoc");
    }
}
```

```

        info.setText("Dodaje urządzenie o podanych informacjach:\n"
                     "Nazwa\n"
                     "Numer IP w formacie 192.168.0.1");
        info.exec();

        return;
    }

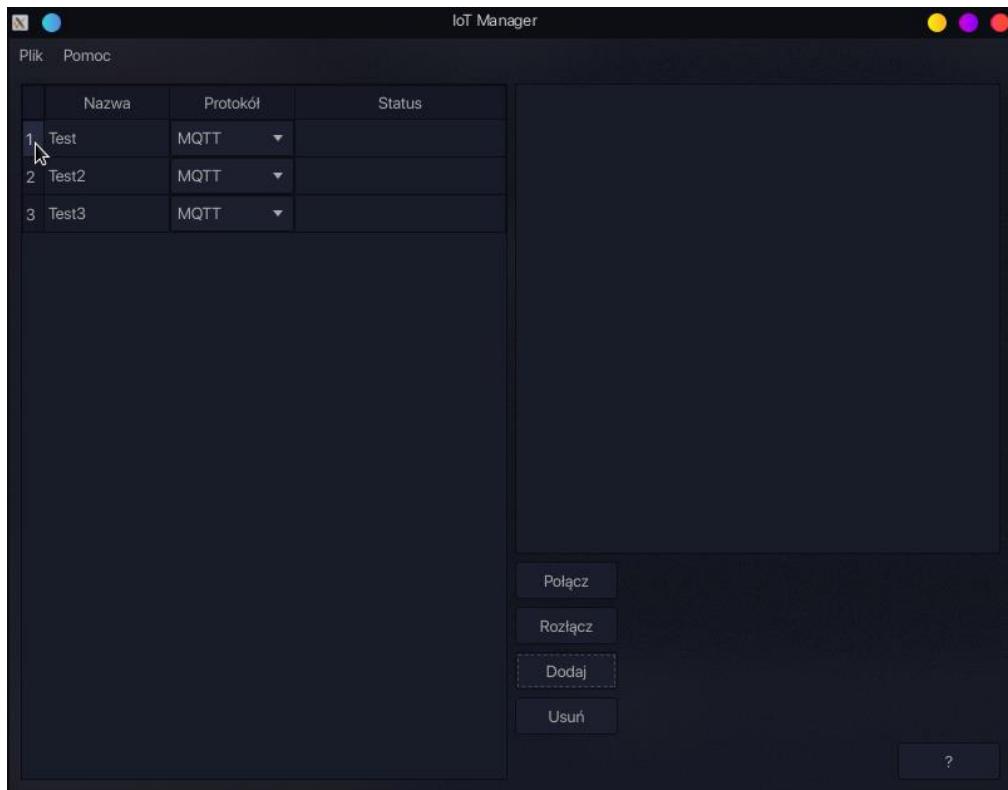
    int ip_ad[4];
    iot* device;
    int index = devices.size();
    bool ok, correct = true;

    QString name = QInputDialog::getText(this, "Dodaj", "Nazwa urządzenia",
                                         QLineEdit::Normal, "", &ok);
    if (ok)
    {
        QString ip = QInputDialog::getText(this, "Dodaj", "Adres IP",
                                         QLineEdit::Normal, "", &ok);
        QRegExp ipFormat("^[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}$");
        if (ok && ipFormat.exactMatch(ip))
        {
            QStringList ipList = ip.split(".");
            ip_ad[0] = ipList.value(0).toInt();
            ip_ad[1] = ipList.value(1).toInt();
            ip_ad[2] = ipList.value(2).toInt();
            ip_ad[3] = ipList.value(3).toInt();
            for (int i = 0; i < 4; i++) if (ip_ad[i] > 255 || ip_ad[i] < 0)
                correct = false;

            if (correct)
            {
                device = new iot(index, name, ip_ad, 0);
                devices.push_back(device);
                display();
            }
        }
    }
}

```

W celu zaznaczenia konkretnego urządzenia w tablicy (potrzebne do operacji łączenia oraz usuwania) należy kliknąć numer wiersza po lewej od nazwy urządzenia, które chcemy zaznaczyć:



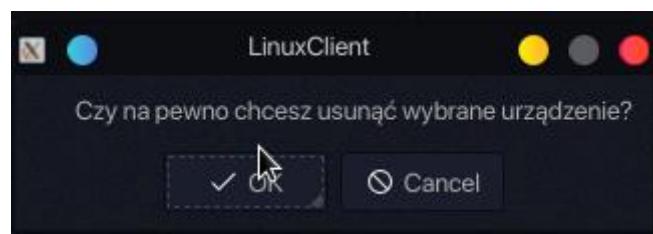
Rysunek 82. Położenie numeru wiersza.

Zaznaczony wiersz zostanie podświetlony:

	Nazwa	Protokół	Status
1	Test	MQTT	
2	Test2	MQTT	
3	Test3	MQTT	

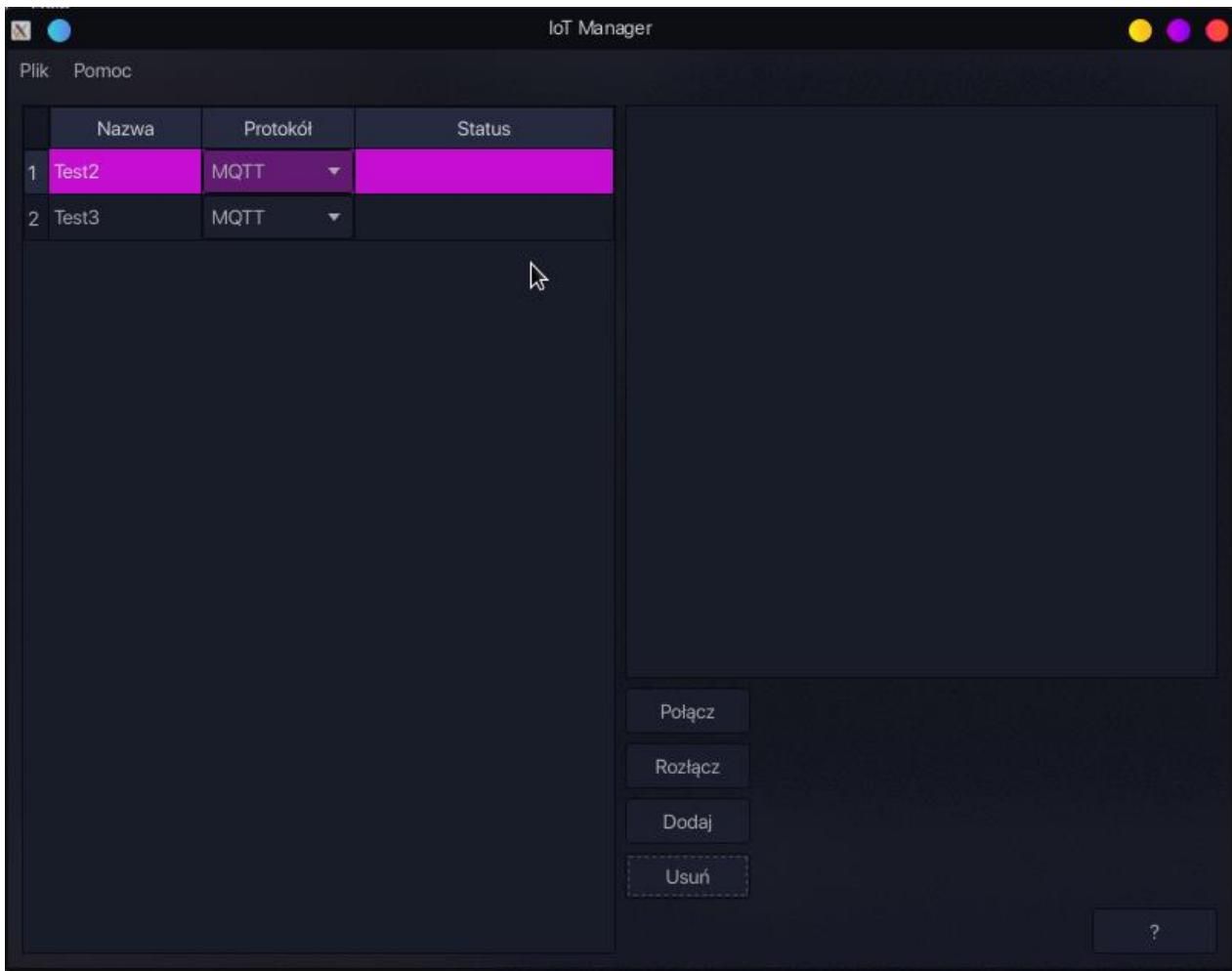
Rysunek 83 .Zawartość tabeli po zaznaczeniu pierwszego wiersza

Po zaznaczeniu wybranego urządzenia można wykonać operację usuwania poprzez wcisnięcie przycisku „Usuń”. Pojawi się prośba o potwierdzenie:



Rysunek 84. Potwierdzenie usunięcia urządzenia

Wygląd okna aplikacji po usunięciu pierwszego urządzenia na liście:



Rysunek 85. Usunięcie urządzenia

```
void UserInterface::on_deleteButton_clicked()
{
    if (helpmode)
    {
        QMessageBox info;
        info.setWindowTitle("Pomoc");
        info.setText("Usuwa wybrane urządzenie z listy");
        info.exec();

        return;
    }

    QModelIndexList selection = ui->iot_table->selectionModel()->selectedRows();

    if (selection.size() > 0)
    {
        QMessageBox confirm;
        confirm.setText("Czy na pewno chcesz usunąć wybrane urządzenie?");
        confirm.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
        int ret = confirm.exec();

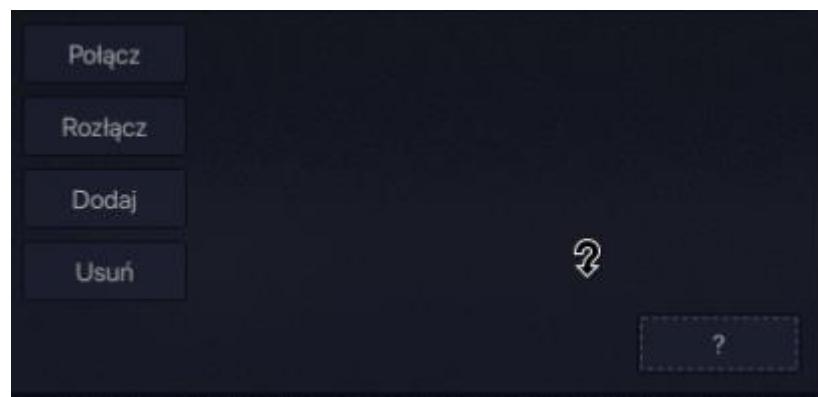
        if (ret == QMessageBox::Ok)
        {
            QModelIndex index = selection.at(0);
            int row = index.row();

            devices.removeAt(row);

            display();
        }
    }
}
```

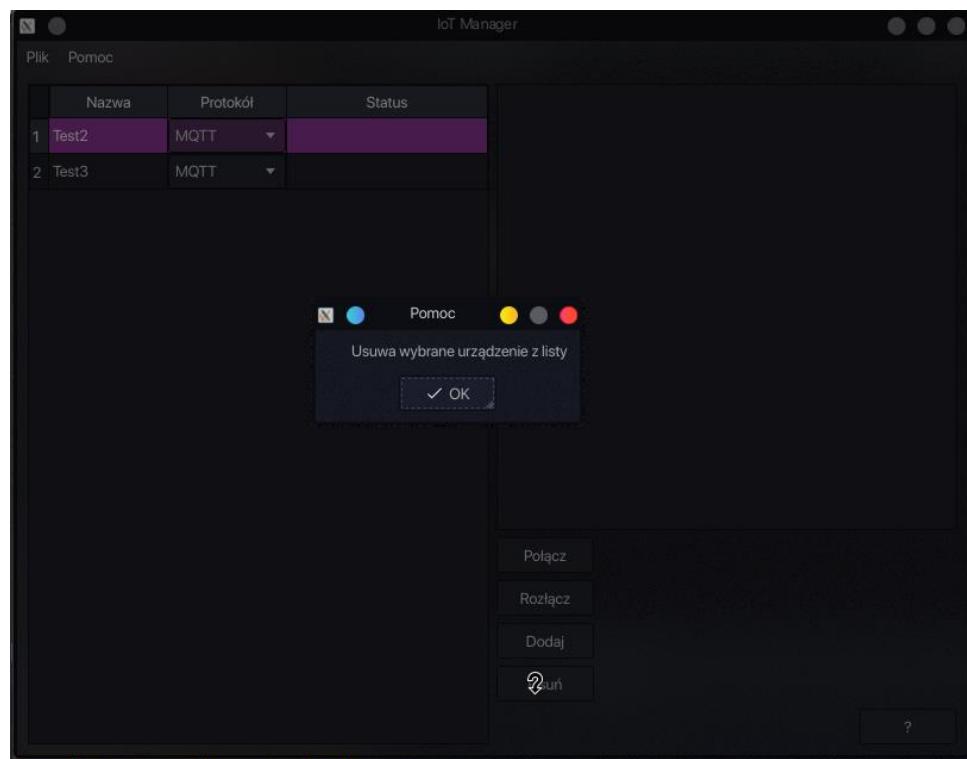
```
        }  
    }  
}
```

Przycisk „?” przełącza w tryb pomocy – widoczne jest to po zmianie kurSORA na znak zapytania, bądź kurSOR ze znakiem zapytania obok.



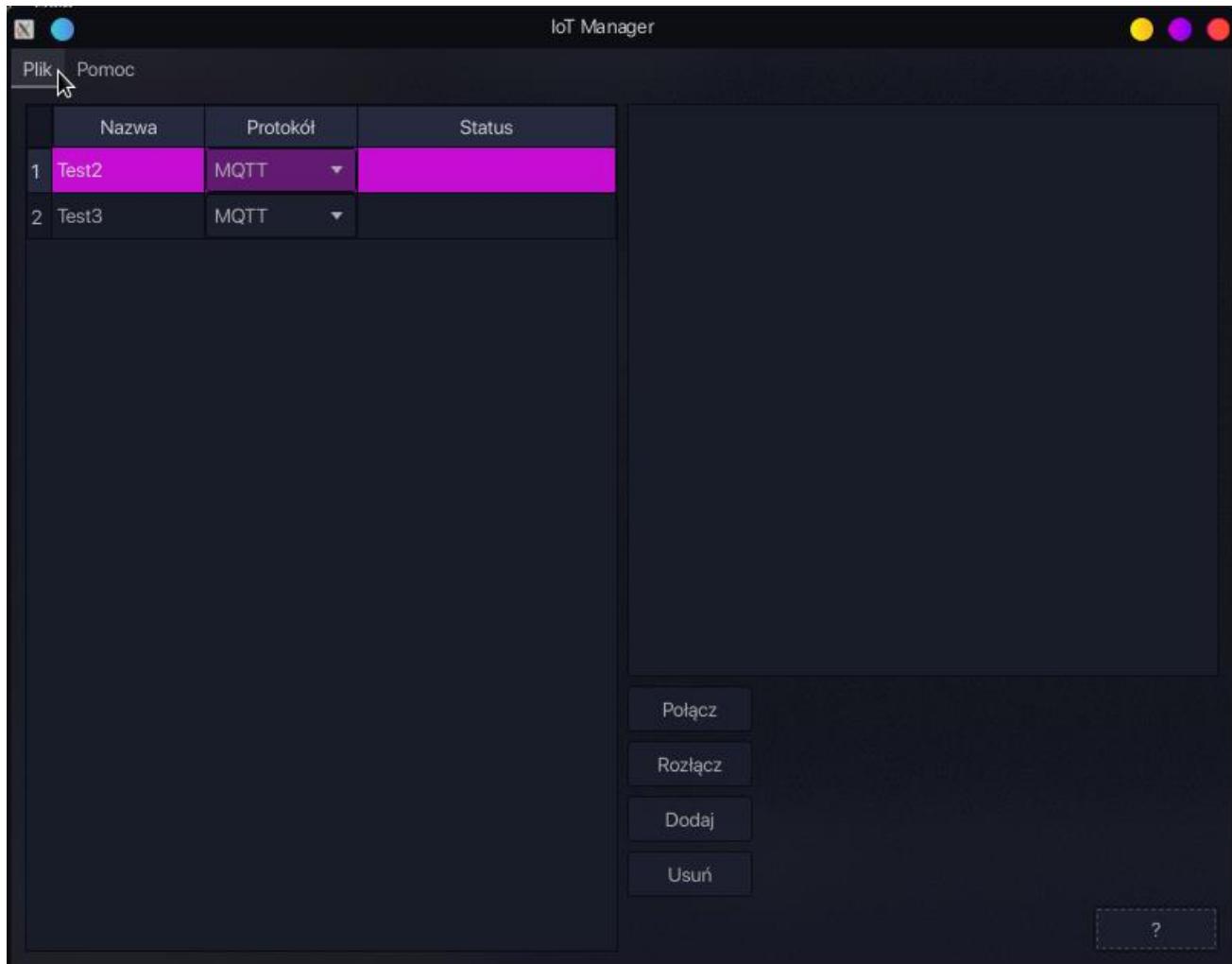
Rysunek 86. Znak zapytania na środku przedstawia przykładowy kurSOR w trybie pomocy

Tryb pomocy sprawia, że użytkownik może sprawdzić, do czego służy dowolny przycisk po naciśnięciu go. Informacje o przycisku zostaną wyświetcone w dodatkowym okienku.



Rysunek 87. Wygląd aplikacji po wcisnięciu przycisku "Usuń" w trybie pomocy

17.4.2. Pasek Menu



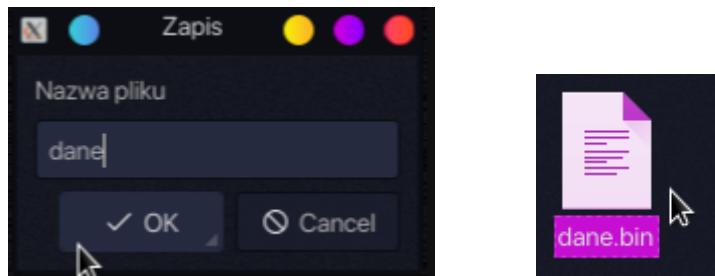
Rysunek 88. Wybór opcji "Plik" w menu

- **Zapis okresowy** – włączenie opcji zapisu okresowego do pliku – wyświetla okno dialogowe, w którym można wprowadzić przedział czasowy, co jaki dokonywany będzie zapis (1-360 minut) a następnie okno dialogowe pozwalające na wprowadzenie nazwy pliku, do którego dane mają być zapisywane.



Rysunek 89. Zapis okresowy

- **Zapisz dane z sensorów** – otwarcie okna dialogowego pozwalającego na wpisanie nazwy pliku, pod jaką chce się zapisać dane



Rysunek 90. Plik z danymi o zapisanych urządzeniach widziany w eksploratorze.

```

void UserInterface::on_actionZapisz_dane_z_sesnor_triggered()
{
    if (helpmode)
    {
        QMessageBox info;
        info.setWindowTitle("Pomoc");
        info.setText("Umożliwia natychmiastowy zapis do pliku o podanej nazwie.\n"
                    "Domyślną nazwą pliku jest 'appdata'");
        info.exec();

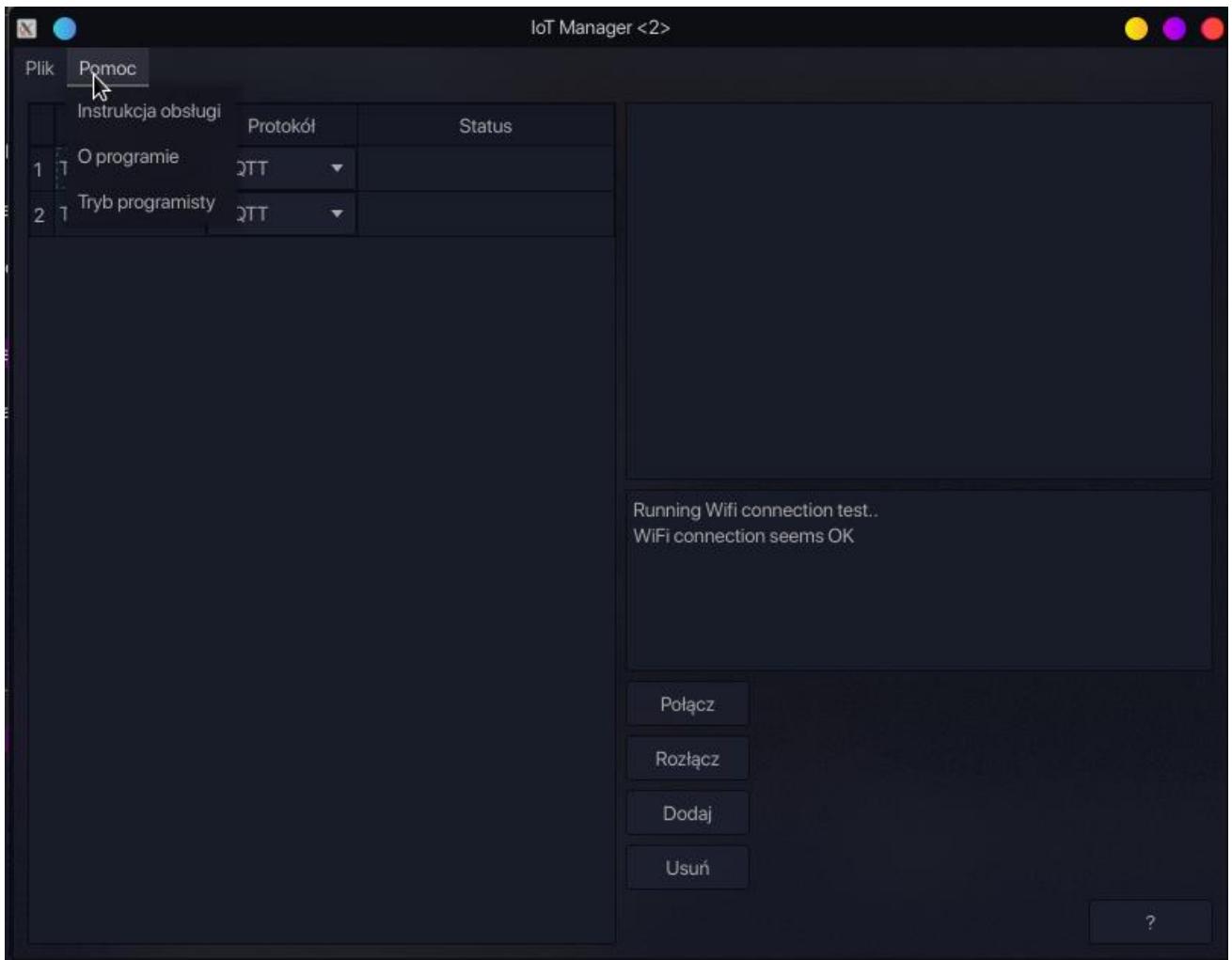
        return;
    }

    bool ok;
    QString fname = QInputDialog::getText(this, "Zapis", "Nazwa pliku", QLineEdit::Normal,
                                         "appdata", &ok);

    if (ok && !fname.isEmpty())
    {
        manager.setfilename(fname);
        manager.savedevices(devices);
    }
}

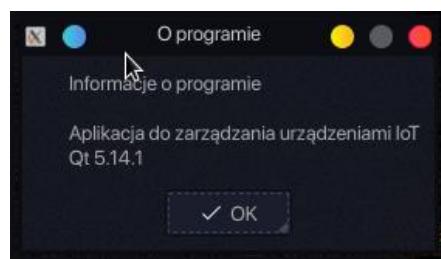
```

- **Zakończ** – kończy pracę programu;



Rysunek 91. Wybór opcji „Pomoc” w menu

- **Instrukcja obsługi** – wyświetla okno zawierające podstawowe instrukcje pomagające zrozumieć obsługę programu;
- **O programie** – wyświetla informacje na temat programu.



Rysunek 92. Informacje o programie

```
void UserInterface::on_actionO_programie_triggered()
{
    if (helpmode)
    {
        QMessageBox info;
        info.setWindowTitle("Pomoc");
        info.setText("Wyświetla informacje o programie");
        info.exec();

        return;
    }

    QMessageBox info;
```

```
        info.setWindowTitle("O programie");
        info.setText("Informacje o programie \n\n"
                     "Aplikacja do zarządzania urządzeniami IoT \n"
                     "Qt 5.14.1");
        info.exec();
    }
```

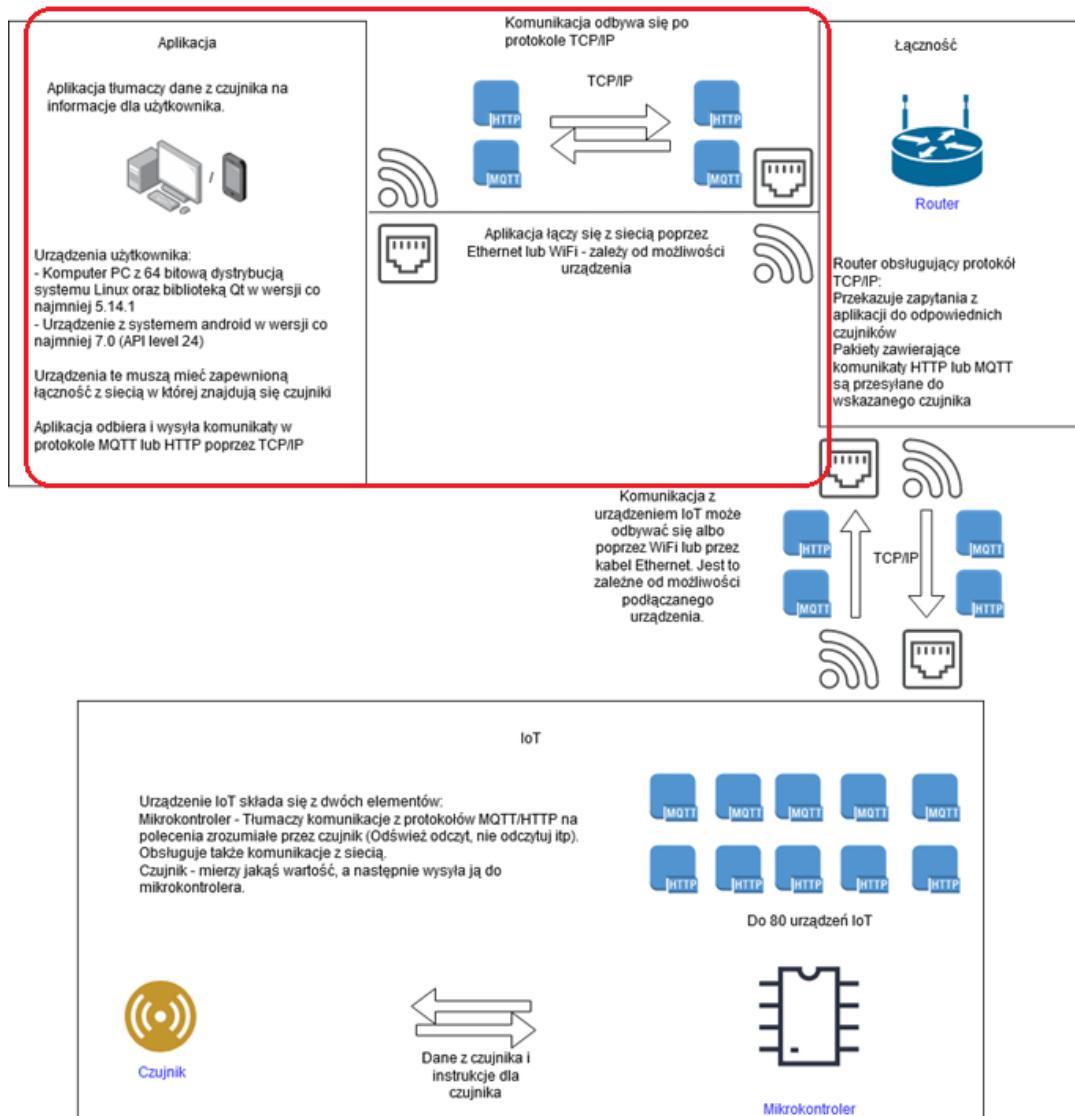
17.5. Uwagi rozwojowe

- Oprócz przycisku „Połącz” w czwartej aktywności na schemacie „Wyświetlanie odczytów” przyda się dodatkowy przycisk „Odczyt”, żeby już po połączeniu nie było potrzeby klikania jeszcze raz w „Połącz” jak chcemy nowy odczyt
- Adres ip można przechowywać w klasie Device jako string zamiast listy, jak wysyłamy zapytanie http to adres i tak jest konwertowany na string- lista rzeczywiście lepsza przy dodawaniu nowych urządzeń bo łatwiej sprawdza się poprawność wprowadzonego adresu - najlepiej byłoby dzielić na listę tylko przy dodawaniu nowego urządzenia a potem już konwertować i operować na stringu

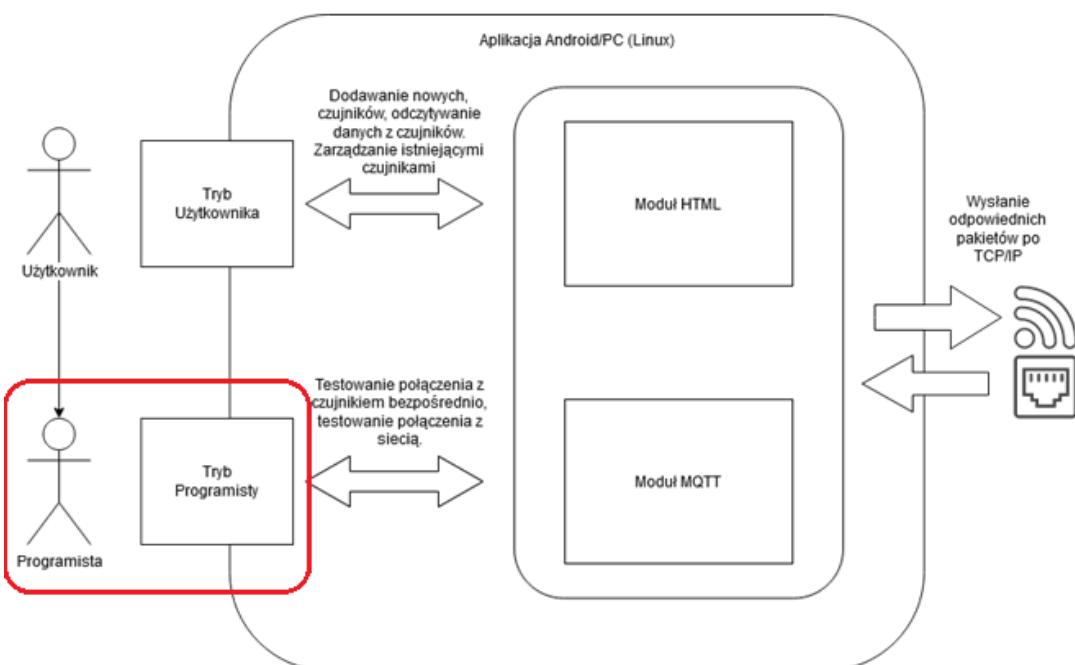
18. Oprogramowanie administratora – Router WiFi (Tryb administratora w aplikacji na system Linux)

18.1. Przeznaczenie

Tryb administratora jest wykorzystywany do dokładniejszej analizy działania programu oraz testów połączenia.



Rysunek 93. Aplikacja na system Linux na schemacie ogólnym



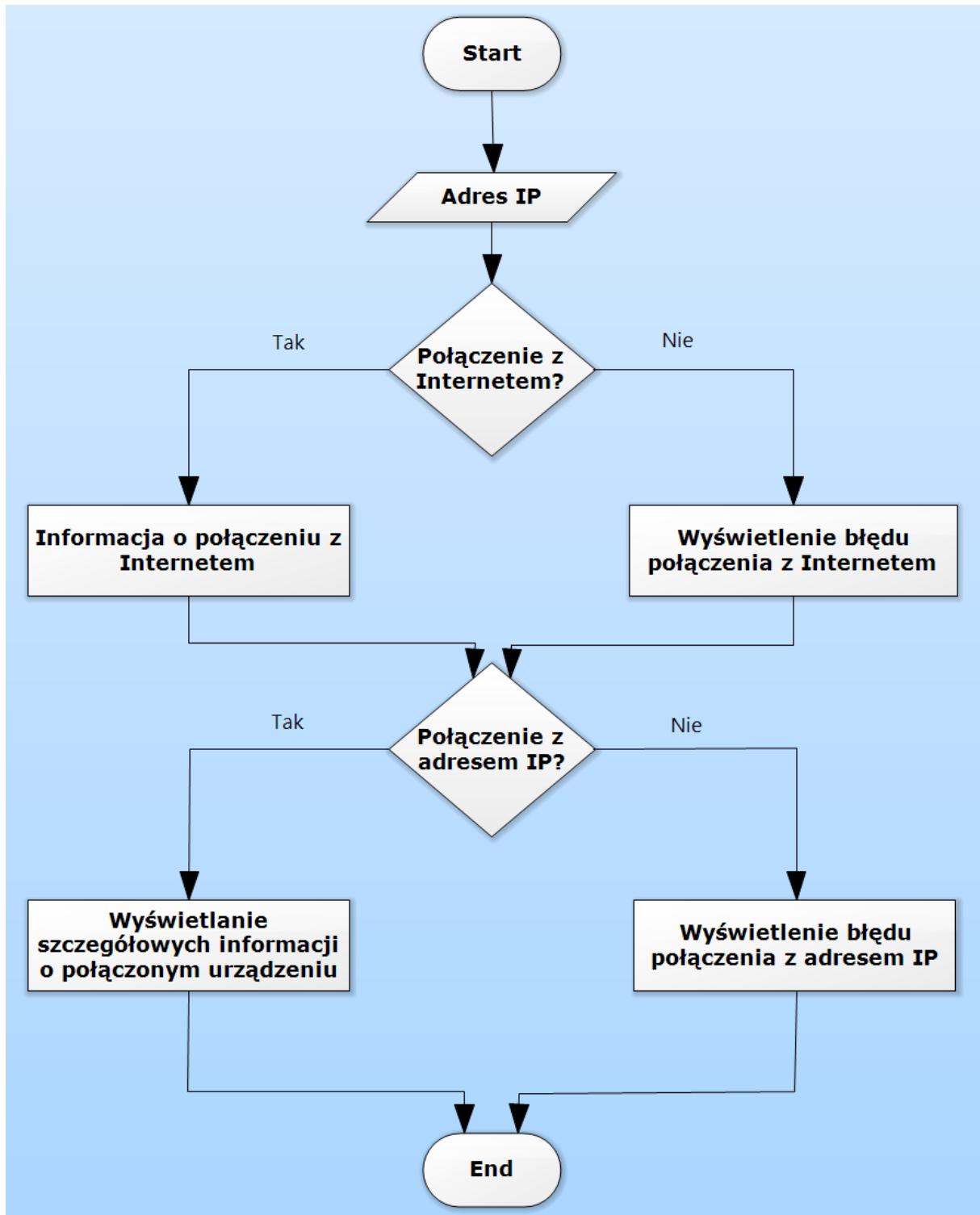
Rysunek 94. Tryb programisty (tryb administratora) na system Linux

18.2. Uruchomienie

Instalacja i uruchomienie aplikacji zostało opisane w rozdziale [16.2](#)

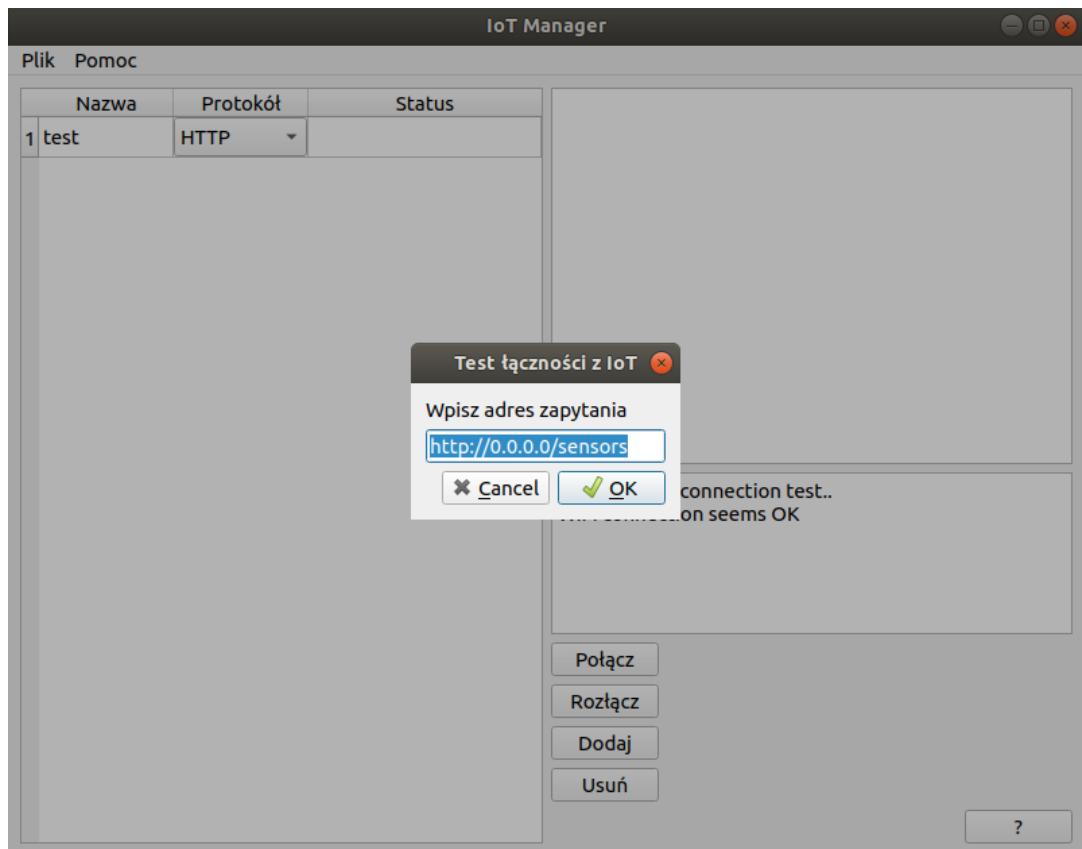
Tryb administratora uruchamiany jest poprzez aplikację desktopową na system Linux przez wybranie opcji Pomoc -> Tryb Programisty.

18.3. Schemat programu



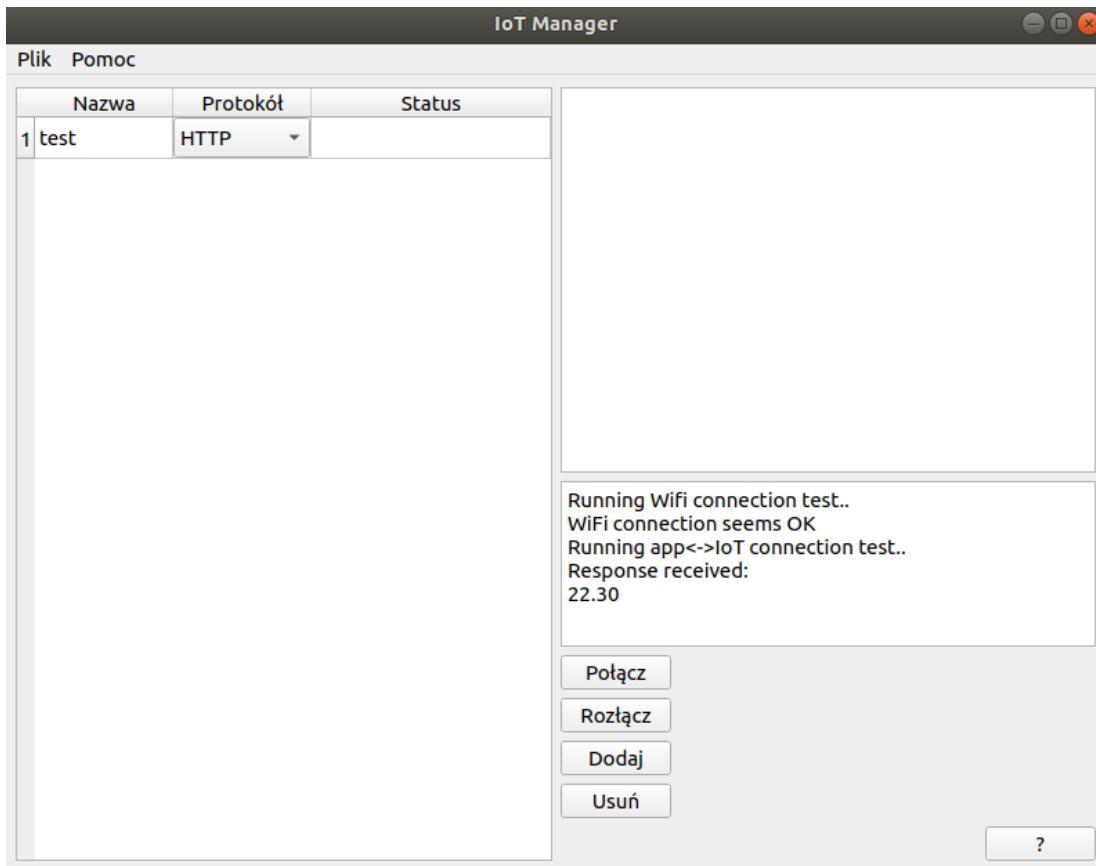
Rysunek 95. Działanie trybu administratora

18.4. Opis oprogramowania i komentarze.



Rysunek 96. Test łączności

Uruchomienie trybu administratora powoduje uruchomienie testów łączności WiFi oraz test bezpośredniej łączności z wpisanyem przez użytkownika adresem urządzenia IoT.



Rysunek 97. Odpowiedź systemu

Ponad przyciskiem „Połącz” pojawia się również dodatkowe pole tekstowe, wyświetlające komunikaty wyjątków oraz dodatkowe informacje.

```
void UserInterface::on_actionTryb_programisty_triggered()
{
    if (helpmode)
    {
        QMessageBox info;
        info.setWindowTitle("Pomoc");
        info.setText("Pozwala przejść w tryb programisty:\n\n"
                    "Sprawdzenie łączności Wi-Fi - automatyczne\n"
                    "Test łączności z ręcznie wpisanym adresem - należy ręcznie\n"
                    "zmienić\n"
                    "domyślne 0.0.0.0 na adres, z którym chcemy sprawdzić łączność");
        info.exec();

        return;
    }

    if (ui->textEdit_2->isVisible())
    {
        ui->textEdit_2->setVisible(false);
    }
    else
    {
        ui->textEdit_2->setVisible(true);

        wifi_connection_test = true;

        ui->textEdit_2->setText("Running Wifi connection test..");

        http.get_request("https://postman-echo.com/get?foo1");
    }
}
```

```

bool iot_test;

QString fname = QInputDialog::getText(this, "Test łączności z IoT",
                                      "Wpisz adres zapytania", QLineEdit::Normal,
                                      "http://0.0.0.0/sensors", &iot_test);

if (iot_test && !fname.isEmpty())
{
    ui->textEdit_2->append("Running app<->IoT connection test..");

    iot_connection_test = true;

    http.get_request(fname);

}
}

```

18.5. Uwagi rozwojowe

- Do trybu programisty należy dodać możliwość wpisywania poleceń przez użytkownika co zówna możliwości aplikacji desktopowej z wersją na systemy android.

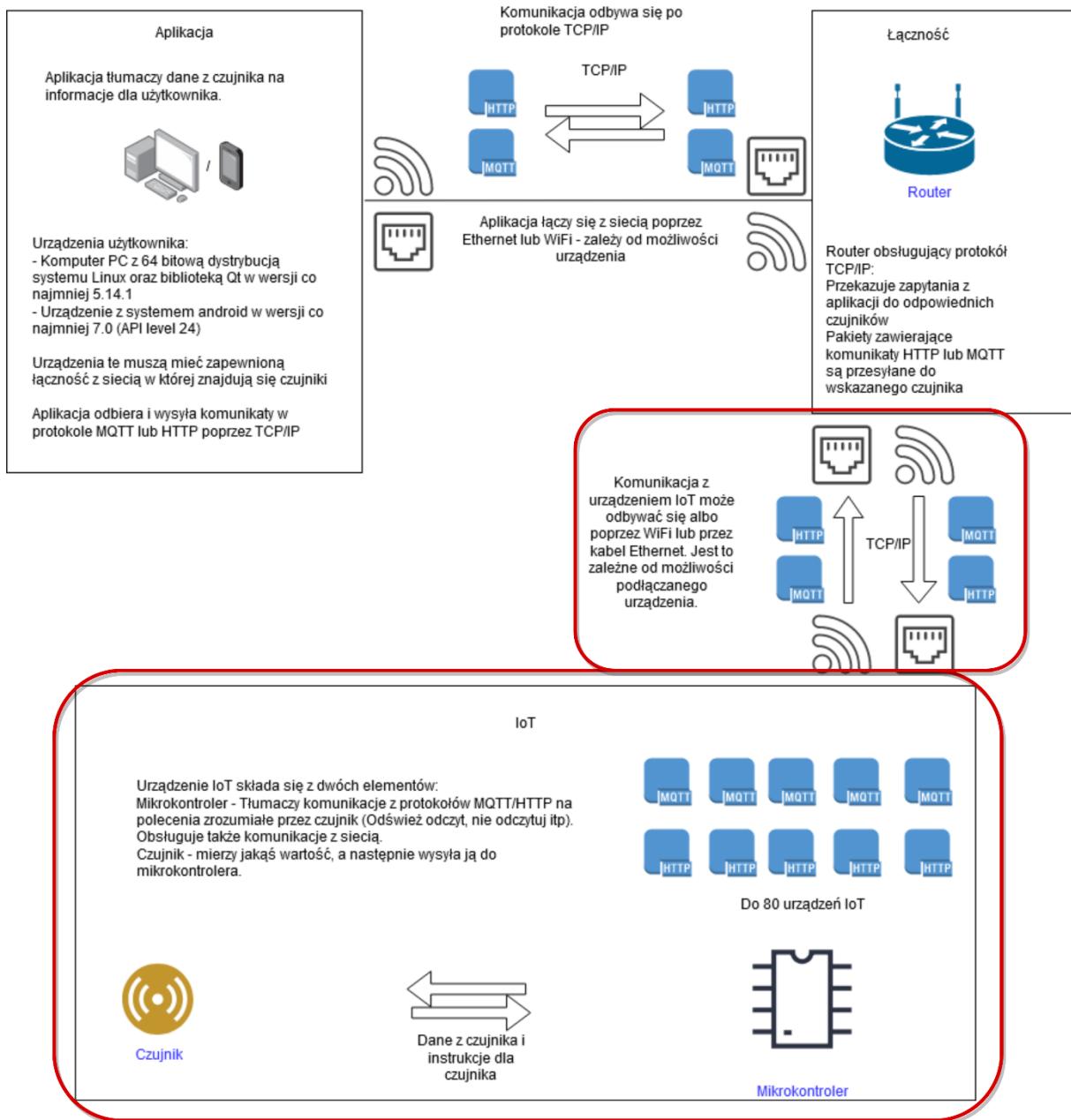
19. Oprogramowanie czujników (urządzeń IoT)

Wykonanie: Mateusz Gurski

Sprawdzenie: Szymon Cichy

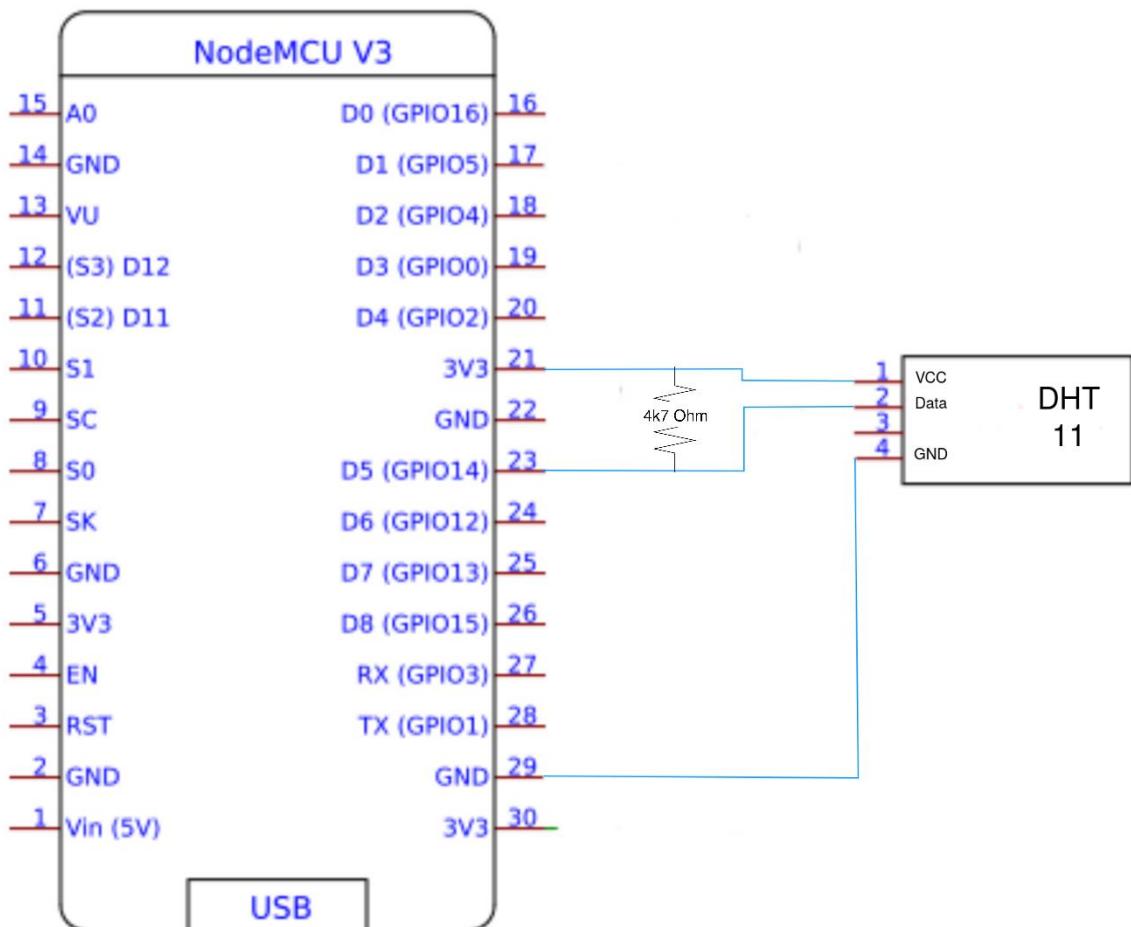
19.1. Przeznaczenie

Urządzenia IoT składają się z dwóch elementów – czujnika i mikrokontrolera. Mikrokontroler odpowiedzialny jest za podłączenie się do sieci, obsługę komunikacji z wykorzystaniem protokołu HTTP/MQTT i obsługę podłączonego czujnika (odbieranie odczytów). Rola tego oprogramowania w projekcie zaznaczona została na ogólnym schemacie czerwonym obramowaniem.

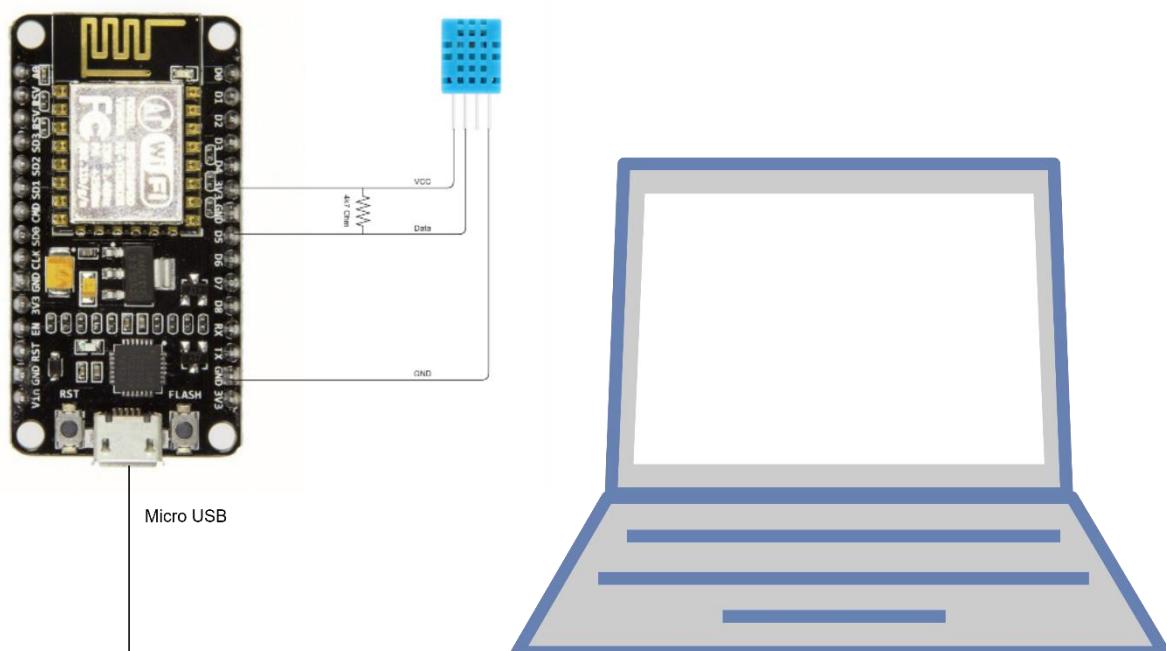


Rysunek 98. Rola oprogramowania czujników (urządzenia IoT) w schemacie ogólnym.

19.2. Schemat połączeń



Rysunek 99. Schemat elektryczny



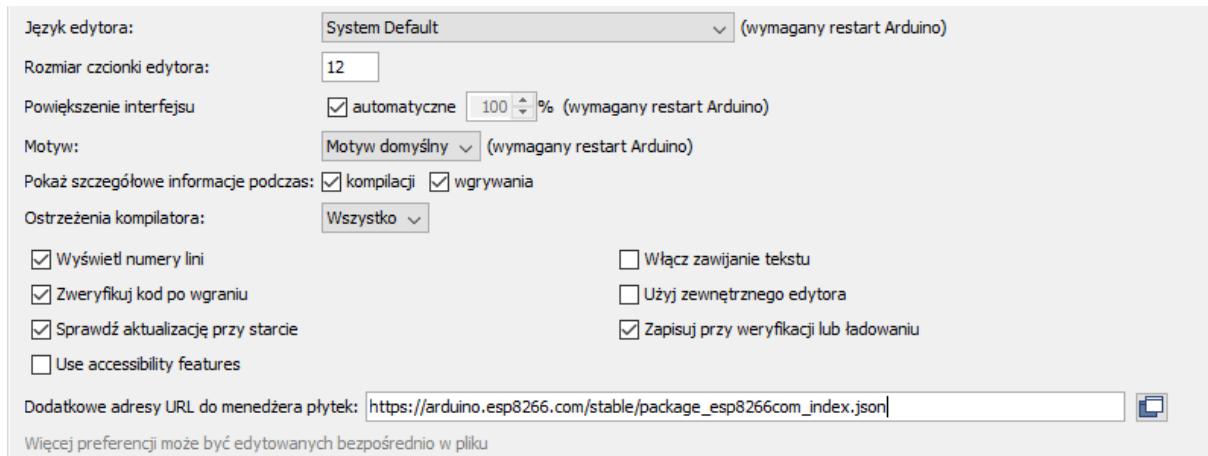
19.3. Rysunek 100. Schemat elektryczno-poglądowy

19.4. Uruchomienie

Programowanie ESP8266 przez Arduino IDE jest obecnie najprostszym i najbezpieczniejszym sposobem programowania tego kontrolera. Aby środowisko poprawnie rozpoznało inny niż kontroler niż Arduino należy pobrać pakiet bibliotek i informacji na temat wybranego przez nas urządzenia.

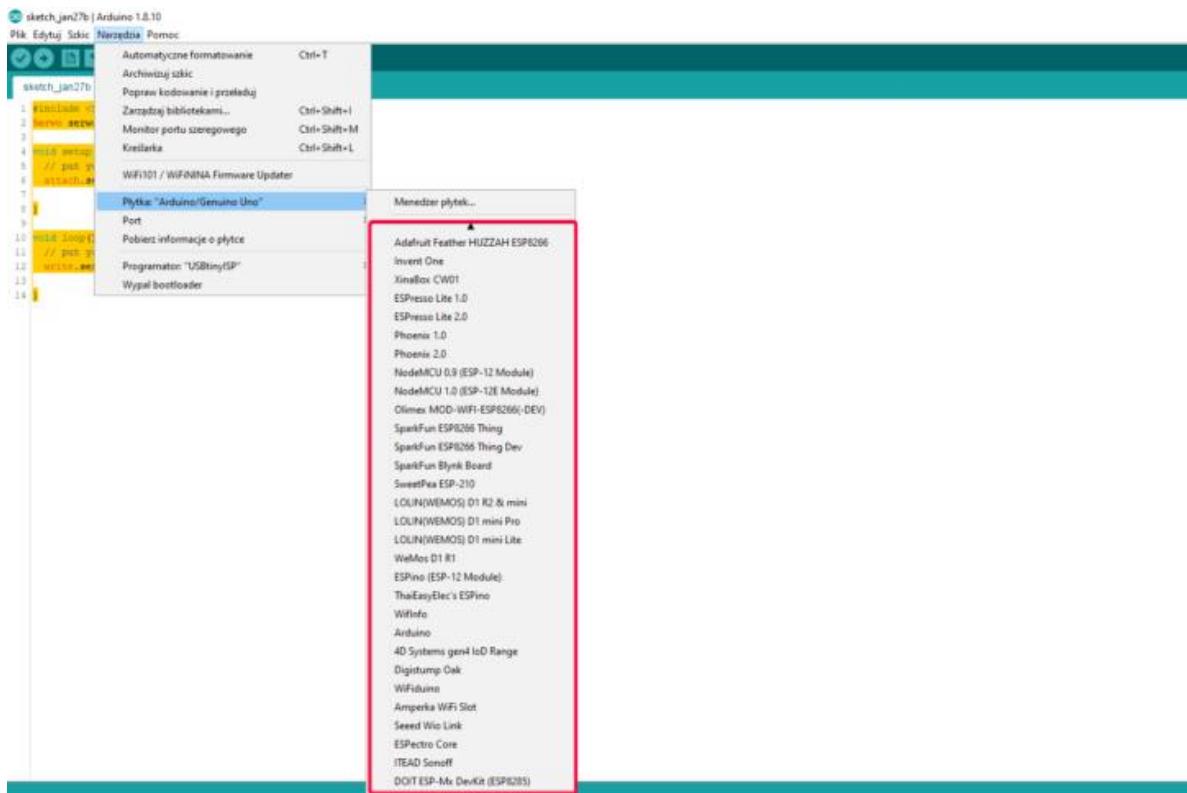
W Arduino IDE wybieramy opcję *Plik > Preferencje* i w polu *Dodatkowe adresy URL do menedżera płytka* wpisujemy poniższy adres:

https://arduino.esp8266.com/stable/package_esp8266com_index.json

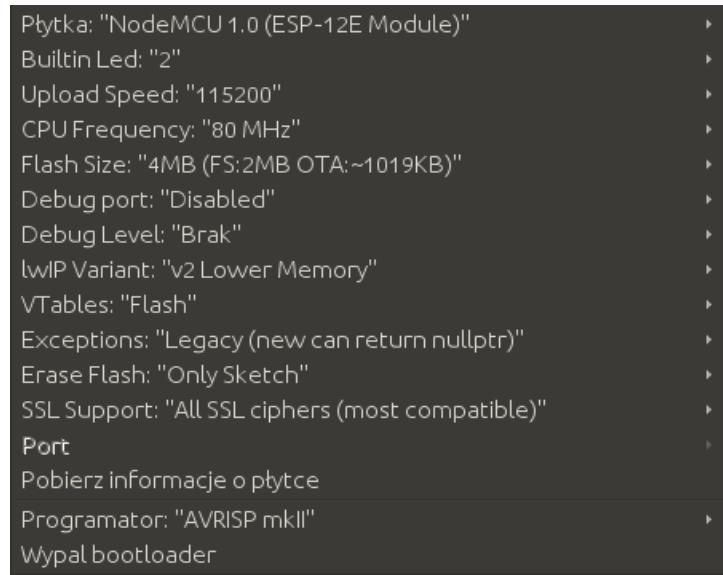


Rysunek 101. Dodanie informacji o ESP8266 do Arduino IDE

W kolejnym kroku wybieramy opcję *Narzędzia > Płytki > Menedżer płytka*, w wyszukiwarce wpisujemy hasło "ESP8266" i instalujemy paczkę nazwaną "esp8266 by ESP8266 Community". Od tej pory podczas wyboru płytka dostępne będą różne modele modułów z ESP8266 na pokładzie.



Rysunek 102. Wybór płytka z ESP w Arduino IDE



Rysunek 103. Preferowane ustawienia

Kolejnym krokiem jest instalacja bibliotek użytych w utworzonym oprogramowaniu

W oprogramowaniu dla wersji HTTP wykorzystane zostały biblioteki **ESPAsyncWebServer**, **ESPAsyncTCP** oraz **DHTesp**. Biblioteki **ESPAsyncWebServer** oraz **ESPAsyncTCP** wykorzystywane są do obsługi żądań HTTP. Nie są one dostępne przez menedżera bibliotek i muszą zostać pobrane ręcznie ze stron:

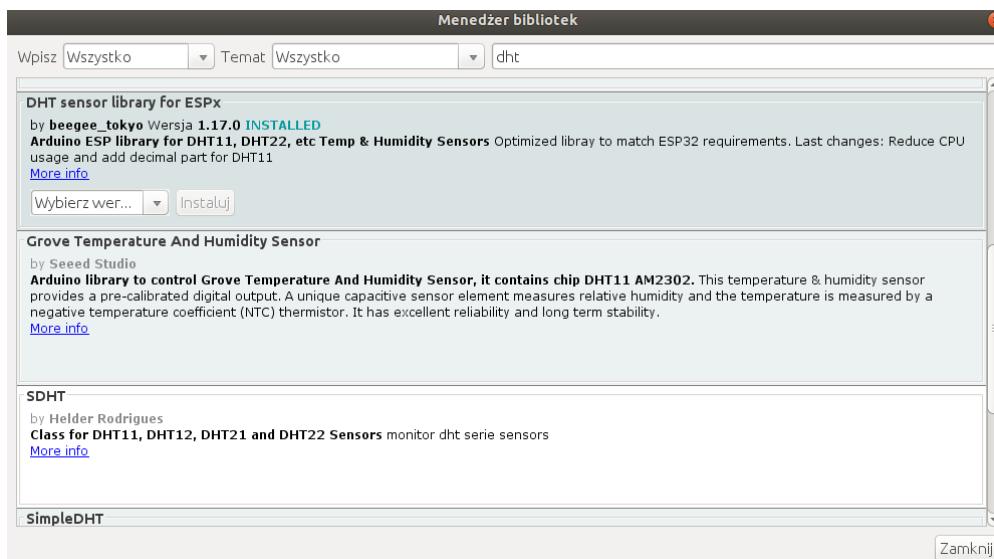
<https://github.com/me-no-dev/ESPAsyncWebServer/archive/master.zip>

oraz

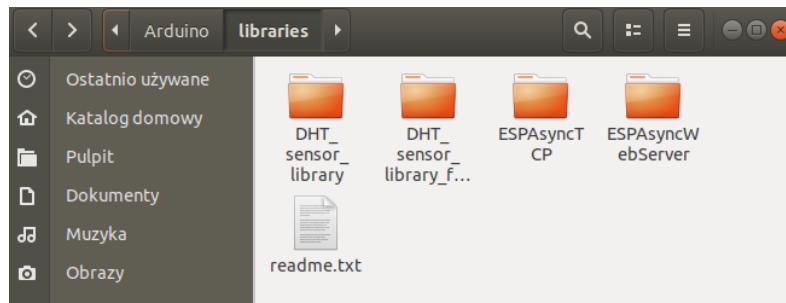
<https://github.com/me-no-dev/ESPAsyncTCP/archive/master.zip>

Obie biblioteki muszą zostać następnie przeniesione do folderu libraries znajdującego się w głównym katalogu programu Arduino IDE.

Bibliotekę odpowiedzialną za obsługę czujnika DHT pobrać można w programie Arduino IDE przy użyciu menedżera bibliotek wpisując hasło dht. Następnie wybieramy bibliotekę **DHT Sensor library for ESPx by beegee_tokyo** i instalujemy ją w najnowszej wersji 1.17.0.



Rysunek 104. Menedżer bibliotek programu Arduino IDE



Rysunek 105. Katalog zawierający potrzebne biblioteki.

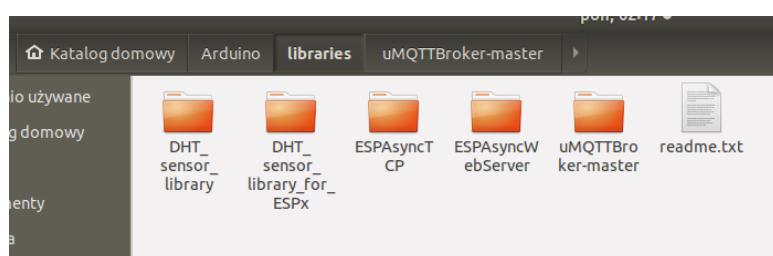
W celu przetestowania protokołu MQTT, należy dodatkowo pobrać bibliotekę uMQTTBroker, dostępną pod adresem:

<https://github.com/martin-ger/uMQTTBroker>

pobrać ją klikając w Clone or download -> Download ZIP:

Rysunek 106. Biblioteka uMQTTBroker

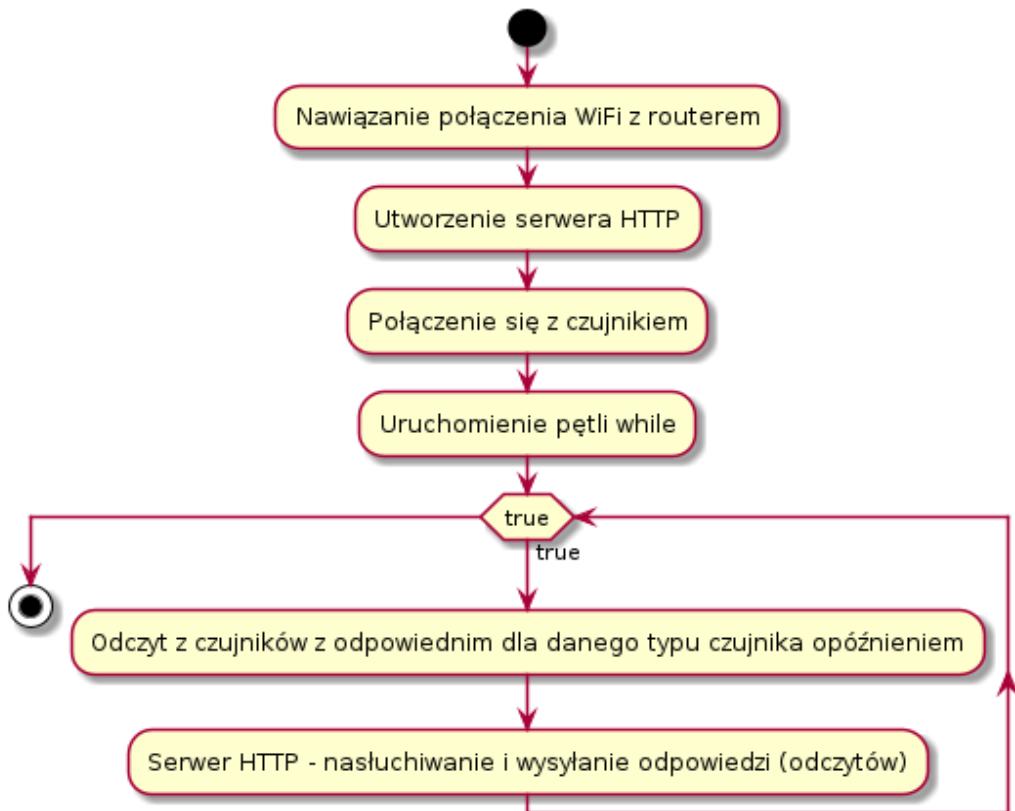
I wypakować do folderu libraries:



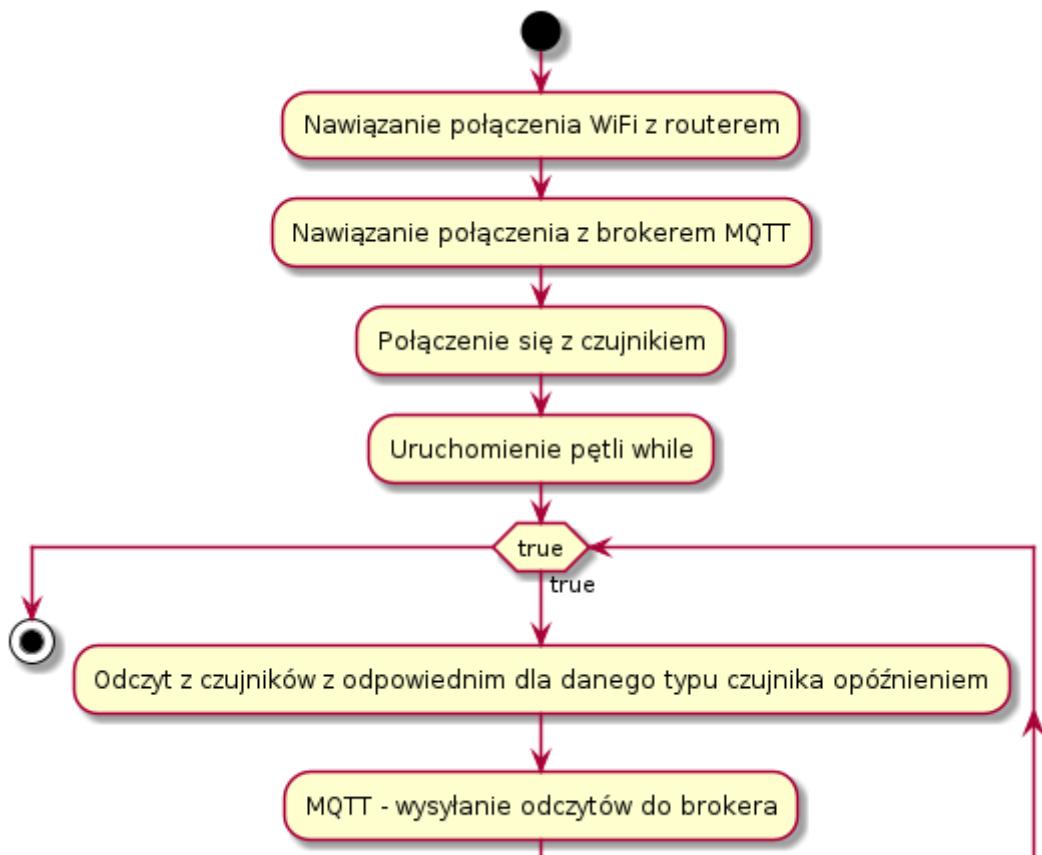
Rysunek 107. Katalog zawierający potrzebne biblioteki

19.4. Schemat programu

Utworzono zostały dwie wersje oprogramowania urządzeń IoT, ze względu na potrzebę przetestowania dwóch różnych protokołów komunikacji. Jedna z wersji obsługuje komunikację HTTP a druga komunikację MQTT. Schematy obu programów przedstawione zostały poniżej.



Rysunek 108. Schemat programu na urządzenie obsługujące protokół http



Rysunek 109. Schemat programu obsługującego protokół MQTT

19.5. Opis oprogramowania z komentarzami

Zgodnie z powyższymi schematami utworzone zostały dwie wersje oprogramowania.

- Wersja HTTP

Dołączanie potrzebnych bibliotek oraz inicjalizacja potrzebnych stałych. W polach ssid i password wpisane powinny zostać ssid i hasło do routera do którego połączyć ma się urządzenie.

```
#include <ESP8266WiFi.h>
#include "ESPAsyncWebServer.h"
#include "DHTesp.h"

#define DHTPIN 14 //D5

DHTesp dht;

const char* ssid = "ssid";
const char* password = "password";

float temperature = 0.0;
float humidity = 0.0;
```

Nawiązania połączenia WiFi z routerem oraz utworzenie serwera na porcie 80. Instrukcje rozpoczynające się od Serial. Wykorzystywane są do komunikacji z monitorem portu szeregowego. W momencie połączenia w monitorze tym wyświetlane jest ip urządzenia IoT a następnie wyświetlane są aktualizacje temperatury i wilgotności w celach demonstracyjnych.

```
AsyncWebServer server(80);

void setup(){
    Serial.begin(115200);
    Serial.println();

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.print("Got IP: ");
    Serial.println(WiFi.localIP());
```

Wykorzystanie biblioteki ESPAsyncWebServer. Serwer nasłuchuje i wysyła odpowiedzi na odpowiednie żądania.

```
server.on("/sensors", HTTP_GET, [](AsyncWebRequest *request){
    request->send_P(200, "text/plain", "temperature:C humidity:%");
});

server.on("/temperature", HTTP_GET, [](AsyncWebRequest *request){
    request->send_P(200, "text/plain", String(temperature).c_str());
});
```

```
server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(humidity).c_str());
});
```

Połączenie się z czujnikiem. Inicjalizacja obiektu dht dla parametrów DHTPIN, w naszym przypadku jest to pin D5 oraz wersji czujnika DHT11. Potem, w następnej linii – uruchomienie serwera http.

```
dht.setup(DHTPIN, DHTesp::DHT11);

server.begin();
}
```

Obsługa czujnika DHT11. Odczyty z czujników powinny odbywać się z co najmniej z pewnym interwałem czasu. Wykorzystywana biblioteka do obsługi czujnika DHT posiada metodę, która zwraca odstęp czasu odpowiedni dla danego typu czujnika, co ten odstęp czasu wyczytywane są z czujnika temperatura i wilgotność i aktualizowane są zmienne temperature i humidity, których wartość wysyłana jest z serwera na żądania /temperature i /humidity.

```
void loop(){
delay(dht.getMinimumSamplingPeriod());

float humi = dht.getHumidity();
float temp = dht.getTemperature();

if(!isnan(temp)){
    temperature = temp;
    humidity = humi;

    Serial.println(humidity, 1);
    Serial.println(temperature, 1);
}
}
```

Po wgraniu skryptu w monitorze portu szeregowego (Narzędzia – Monitor portu szeregowego) odczytujemy IP urządzenia IoT. Działanie oprogramowania można przetestować przy użyciu przeglądarki, wpisując do niej adres IP urządzenia oraz /temperaturę.



Rysunek 110. Odczyt temperatury

- Wersja MQTT

Podobnie jak w przypadku HTTP - dołączanie potrzebnych bibliotek oraz inicjalizacja potrzebnych stałych. W polach ssid i password wpisane powinny zostać ssid i hasło do routera do którego połączyć ma się urządzenie.

```
#include <ESP8266WiFi.h>
#include "uMQTTBroker.h"
#include "DHTesp.h"

#define DHTPIN 14 //D5
```

```
DHTesp dht;

const char* ssid = "ssid";
const char* password = "password";

float temperature = 0.0;
float humidity = 0.0;
```

Nadpisujemy metody klasy uMQTTBroker należącej do biblioteki uMQTTBroker i tworzymy obiekt brokeru MQTT.

//Nadpisanie metod klasy uMQTTBroker

```
class myMQTTBroker: public uMQTTBroker
{
public:
    virtual bool onConnect(IPAddress addr, uint16_t client_count) {
        Serial.println(addr.toString() + " connected");
        return true;
    }

    virtual bool onAuth(String username, String password) {
        Serial.println("Username/Password: " + username + "/" + password);
        return true;
    }

    virtual void onData(String topic, const char *data, uint32_t length) {
        char data_str[length+1];
        os_memcpy(data_str, data, length);
        data_str[length] = '\0';

        Serial.println("received topic " + topic + " with data " + (String)data_str + "");
    }
};
```

myMQTTBroker myBroker;

Pętla setup, wygląda podobnie jak w przypadku wersji HTTP – z tą różnicą, że dodatkowo jest też inicjalizowany broker MQTT.

```
void setup()
{
    Serial.begin(115200);
    Serial.println();

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected..!");
    Serial.print("Got IP: "); Serial.println(WiFi.localIP());

    Serial.println("Starting MQTT broker");
    myBroker.init();

    myBroker.subscribe("#");
```

```
dht.setup(DHTPIN, DHTesp::DHT11);
```

```
}
```

W pętli loop zostało dodatkowo dodane publikowanie odczytów na tematach „broker/humidity” i „broker/temperature”.

```
void loop()
{
    delay(dht.getMinimumSamplingPeriod());

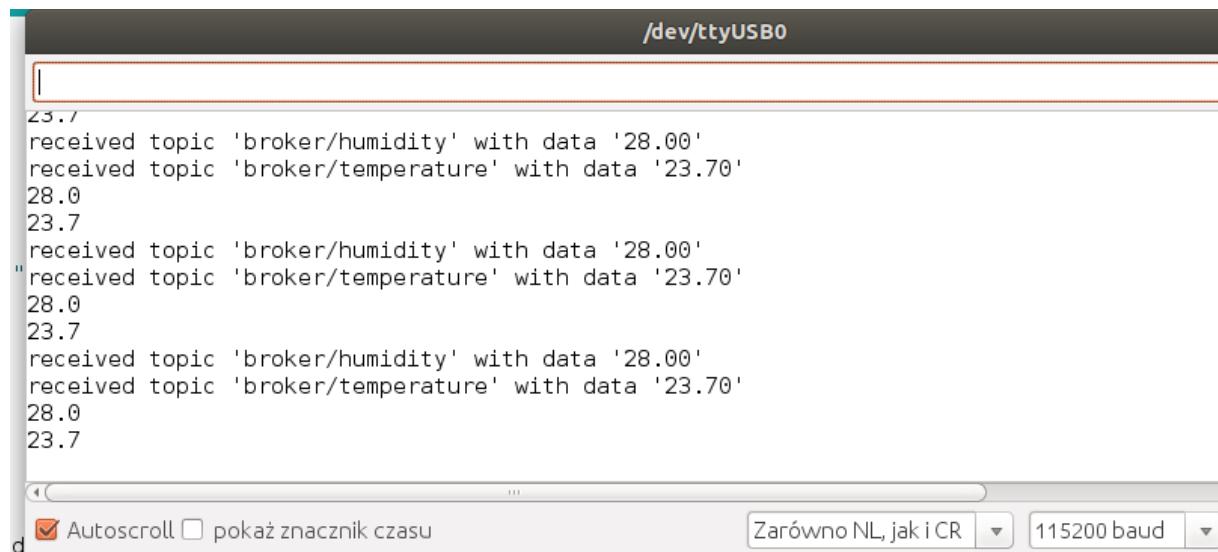
    float humi = dht.getHumidity();
    float temp = dht.getTemperature();

    if(!isnan(temp)){
        temperature = temp;
        humidity = humi;

        myBroker.publish("broker/humidity", (String)humidity);
        myBroker.publish("broker/temperature", (String)temperature);

        Serial.println(humidity, 1);
        Serial.println(temperature, 1);
    }
}
```

Po wgraniu na urządzenie oprogramowania w wersji MQTT, w monitorze portu szeregowego możemy zaobserwować działanie:



Rysunek 111. Monitor portu szeregowego - Test MQTT

19.6. Uwagi rozwojowe

Połączenie się z czujnikiem i jego obsługa wymaga podania w kodzie programu jego konkretnej wersji (`dht.setup(DHTPIN, DHTesp::DHT11)`) co w przypadku zmiany wykorzystywanego typu czujnika DHT, np. z wersji 11 na wersje 22, wymaga też wprowadzenia drobnej zmiany w kodzie oprogramowania czujników. (z `dht.setup(DHTPIN, DHTesp::DHT11)` na `dht.setup(DHTPIN, DHTesp::DHT22)`).

20. Wsparcie użytkownika

Autor Adam Krizar

Aplikacja umożliwia użytkownikowi kontakt ze wsparciem technicznym poprzez wiadomości e-mail lub telefonicznie zależy to wyłącznie od osób dostarczających to wsparcie. Aplikacja oferuje w menu pomocy dane kontaktowe do specjalisty, który jest w stanie przeprowadzić użytkownika przez kolejne kroki diagnostyczne w celu rozwiązania problemu.

20.1. Tryb programistyczny

Zarówno aplikacja na urządzenia z systemem android jak i wersja desktopowa posiadają specjalny tryb programistyczny. Umożliwia on przejrzenie logów aplikacji – informacje o przechwyconych błędach lub podejrzeć dane wymienianie pomiędzy aplikacją, a urządzeniem IoT.

Dodatkową funkcjonalnością jest możliwość pominięcia większości aplikacji i wysłania ręczne zapytania na wskazany adres. Umożliwia to przetestowanie połączenia sieciowego i ograniczenie problemu do samej aplikacji lub reszty systemu.

20.2. Użytkownik czujników

Układy IoT z czujnikami programowane i testowane są przy pomocy wieloplatformowej aplikacji Arduino IDE, która dostępna jest do pobrania na stronie:

<https://www.arduino.cc/en/Main/Software>

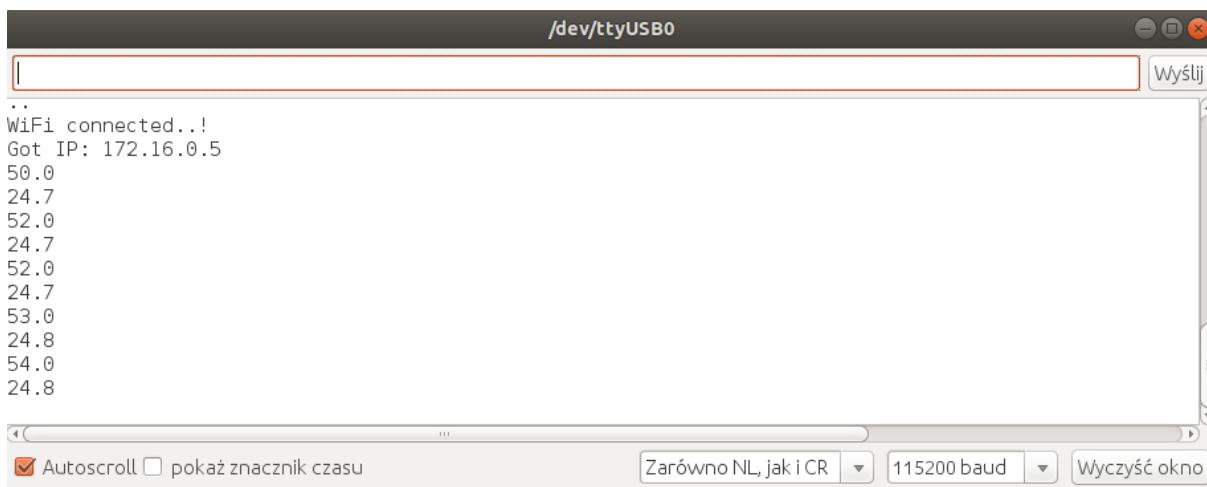
Dostępne są wersje na systemy Windows, Linux oraz MacOs.

Wbudowany monitor portu szeregowego pozwala na podgląd działania urządzeń IoT i bezpośredniego odbierania odczytów z czujników bez potrzeby użycia utworzonych aplikacji.

Monitor portu szeregowego uruchomić można już po podłączeniu do wejścia USB urządzenia IoT.

Narzędzia -> Monitor portu szeregowego.

Na rysunku poniżej prezentacja odczytów. Na początku działania, urządzenie IoT wysyła informację na temat swojego adresu IP a następnie wysyłane są kolejne odczyty wilgotności i temperatury.



The screenshot shows the Arduino IDE Serial Monitor window. The title bar reads '/dev/ttyUSB0'. The main area displays sensor data: 'WiFi connected...!', 'Got IP: 172.16.0.5', followed by a series of temperature and humidity values: 50.0, 24.7, 52.0, 24.7, 52.0, 24.7, 53.0, 24.8, 54.0, 24.8. At the bottom, there are several buttons: 'Autoscroll' (checked), 'pokaż znacznik czasu' (unchecked), 'Zarówno NL, jak i CR' (dropdown), '115200 baud' (dropdown), and 'Wyczyść okno'.

Rysunek 72. Monitorowanie czujników w programie Arduino IDE

21. Kosztorys

Autor Adam Krizar

Głównym kosztem w realizacji naszego projektu są urządzenia IoT konieczne do testowania i prezentacji możliwości naszej aplikacji. Potrzebne są nam dwie platformy testowe:

Mikrokontroler ESP8266: <https://allegro.pl/oferta/esp8266-nodemcu-v3-wifi-2-4ghz-ch340-do-arduino-7241549772>, koszt 18,90 zł.

Czujnik DHT11: <https://allegro.pl/oferta/dht11-czujnik-temperatury-i-wilgotnosci-arduino-7487941486>, koszt 4,70 zł

Całkowity koszt w zależności od wybranej podstawki wynosi odpowiednio:

ESP8266: 47,20 zł

22. Plan realizacji

- Pierwszy punkt kontrolny [19.03]**

Implementacja prototypowej wersji aplikacji na system Linux. Zaimplementowanie protokołu http po stronie aplikacji.

- Drugi punkt kontrolny [02.04]**

Rozwój aplikacji na system Linux. Przygotowanie pierwszego urządzenia IoT i przetestowanie działania protokołu HTTP. Implementacja protokołu MQTT (bez testów).

- Trzeci punkt kontrolny [23.04]**

Przeniesie aplikacji na system android. Przygotowanie drugiego urządzenia IoT oraz przetestowanie protokołu MQTT.

- Instalacja [07.05]**

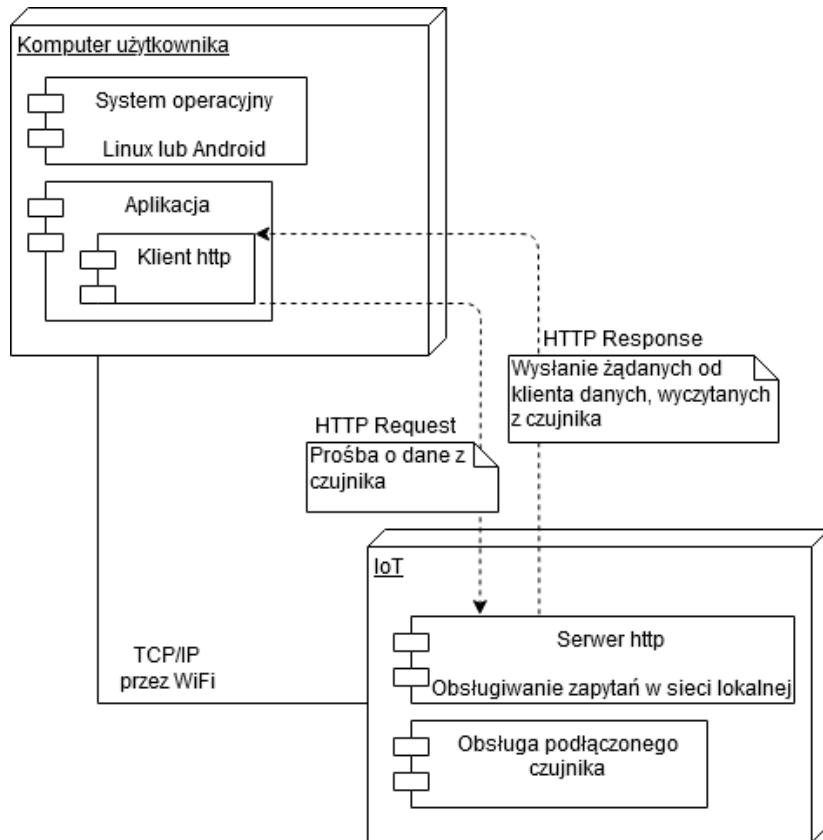
- Testy użytkownika [21.05]**

Wprowadzenie ewentualnych korekt w projekcie interfejsu użytkownika zgodnie z uwagami użytkownika końcowego.

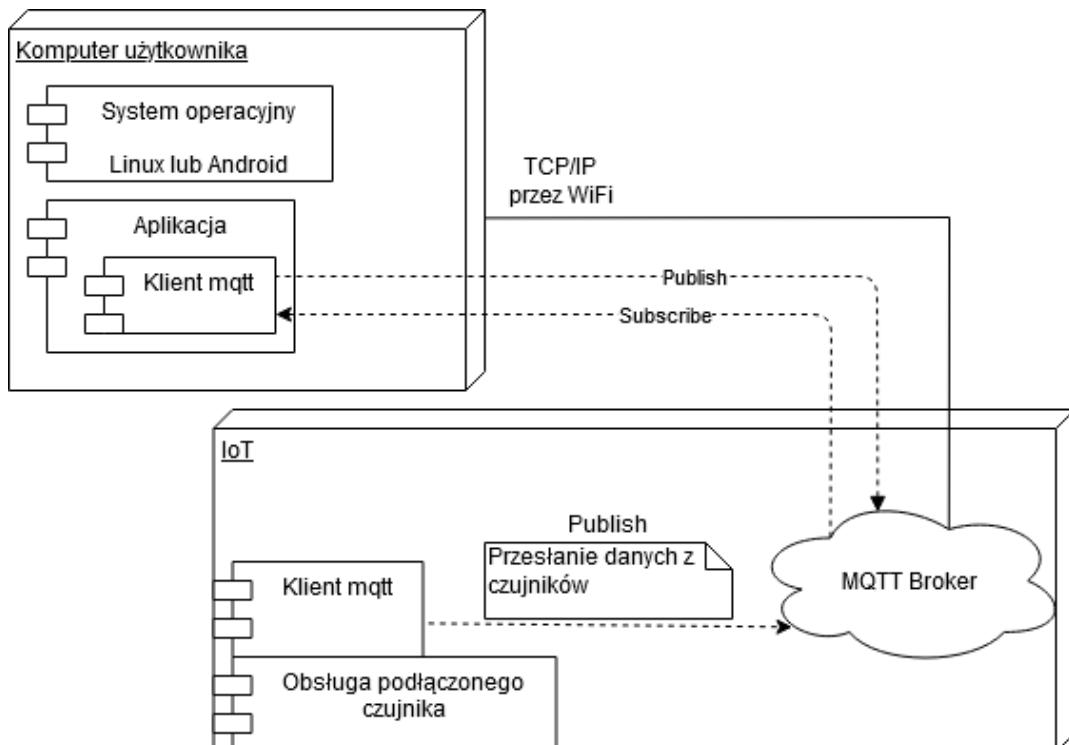
- Oddanie projektu do użytku [04.06]**

- Prezentacja naszych osiągnięć [10.06]**

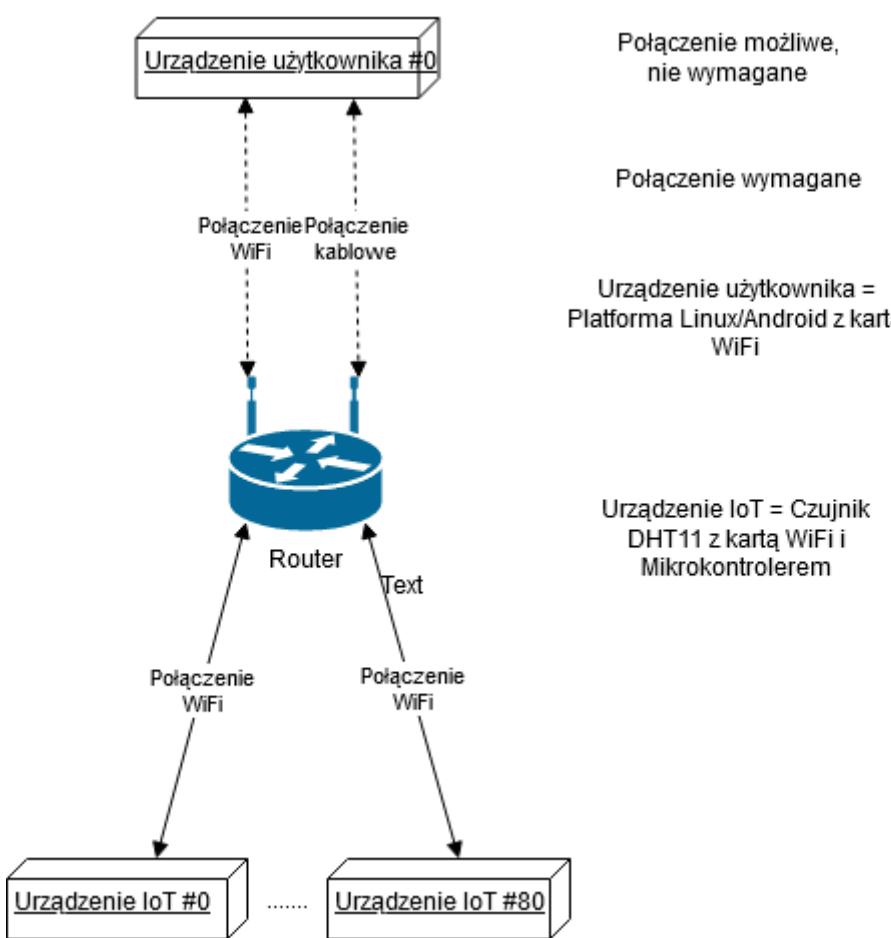
23. Propozycja rozwoju systemu



Rysunek 73. Obsługa protokołu HTTP



Rysunek 74. Obsługa protokołu MQTT



Rysunek 75. Ogólna propozycja użycia aplikacji

24. Źródła

<https://store.arduino.cc/arduino-nano>

<https://www.qt.io/>

<https://store.arduino.cc/arduino-uno-rev3>

<https://learn.adafruit.com/dht>

https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf

<https://en.wikipedia.org/wiki/ESP32>

<https://en.wikipedia.org/wiki/ESP8266>