Politechnika
Wrocławska

Projekt zespołowy

Kierunek	*Informatyka*	**Termin**	*Czwartek 14:15*
Temat	*Projekt elastycznej aplikacji do zarządzania urządzeniami IoT w oparciu o bibliotekę QT*	**Zgłaszający**	*InterElcom*
Skład grupy	*Adam Krizar 241276* *Katarzyna Czajkowska 242079* *Mateusz Gurski 242089* *Szymon Cichy 235093* *Arkadiusz Cichy 236011*	**Nr grupy**	-
Prowadzący	*Dr inż. Jan Nikodem*	**data**	*23 kwietnia 2020*

Spis treści

1. Plan zadań

Wykonanie: Grupa
Sprawdził: Adam Krizar

Wykonawca: Adam Krizar

- Korekta dokumentacji i dodawanie nowych elementów
- Zatwierdzanie prac.

- **Grupa 1**

Wykonawca: Mateusz Gurski
Sprawdzenie: Arkadiusz Cichy

- Implementacja oprogramowania na wybrane urządzenie IoT do obsługi protokołu HTTP oraz wybranego czujnika. Program techniczny obsługi dla IoT - Instrukcja wgrywania utworzonego oprogramowania wraz z opisem użytych bibliotek oraz listingiem kodu wraz z komentarzami.

- Implementacja obsługi protokołu HTTP w aplikacji desktopowej. Opis działania, użytych bibliotek oraz listing najważniejszych fragmentów kodu wraz z komentarzami

- **Grupa 2**

Wykonawca: Arkadiusz Cichy
Sprawdzenie: Szymon Cichy

- Implementacja obsługi protokołu MQTT w aplikacji desktopowej. Opis działania, użytych bibliotek oraz listing najważniejszych fragmentów kodu wraz z komentarzami
- Implementacja oprogramowania na wybrane urządzenie IoT do obsługi protokołu MQTT oraz wybranego czujnika. Program techniczny obsługi dla IoT - Instrukcja wgrywania utworzonego oprogramowania wraz z opisem użytych bibliotek oraz listingiem kodu wraz z komentarzami.

- **Grupa 3**

Wykonawca: Szymon Cichy/Adam Krizar
Sprawdzenie: Katarzyna Czajkowska

- Implementacja interfejsu w aplikacji android oraz opis użytych funkcji do stworzenia projektu.
- Implementacja komunikacji z urządzeniem IoT umożliwiających odbieranie prostych komunikatów wraz z opisem użytych bibliotek..

- **Grupa 4**

Wykonawca: Katarzyna Czajkowska
Sprawdzenie: Mateusz Gurski

- Dopracowanie interfejsu graficznego (obsługa przycisków, dodawanie nowego urządzenia, okno pomocy)
- Opis interfejsu użytkownika (zrzuty ekranu, instrukcja obsługi, wykorzystane biblioteki)

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.
- 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 mikrontrolera, 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 aplikacji działającej minimum na dwie platformy (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 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.

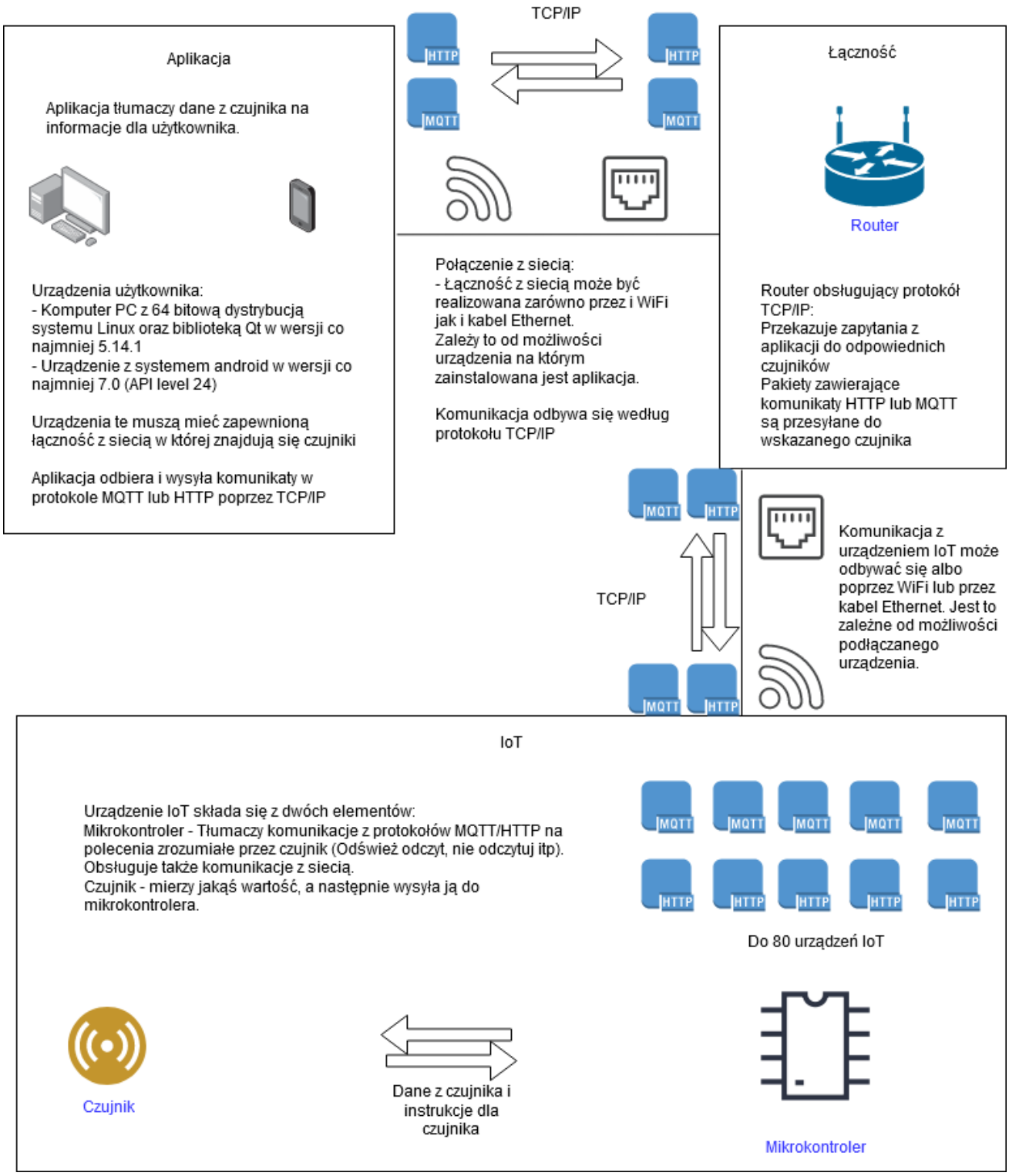
- Wymaganie implementacji minimum dwóch protokołów komunikacji z IoT: Implementacja protokołu HTTP oraz MQTT w naszej aplikacji oraz w testowym urządzeniu IoT. Te dwa protokoły zostały wyszczególnione jako przykładowe przez zgłaszającego oraz należą do najpopularniejszych rozwiązań na rynku co zapewni 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.



Rysunek 1. Ogólny schemat

5. Środowisko

Wykonanie: Szymon Cichy
Sprawdzenie: Adam Krizar

5.1. 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.

5.2. 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 to 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 jeżyka 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.

6. Wybrane urządzenia/czujniki

Wykonanie: Arkadiusz Cichy
Sprawdzenie: Szymon Cichy

6.1. 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 potrzebnego na zbudowanie działającego układu.

6.2. 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**:

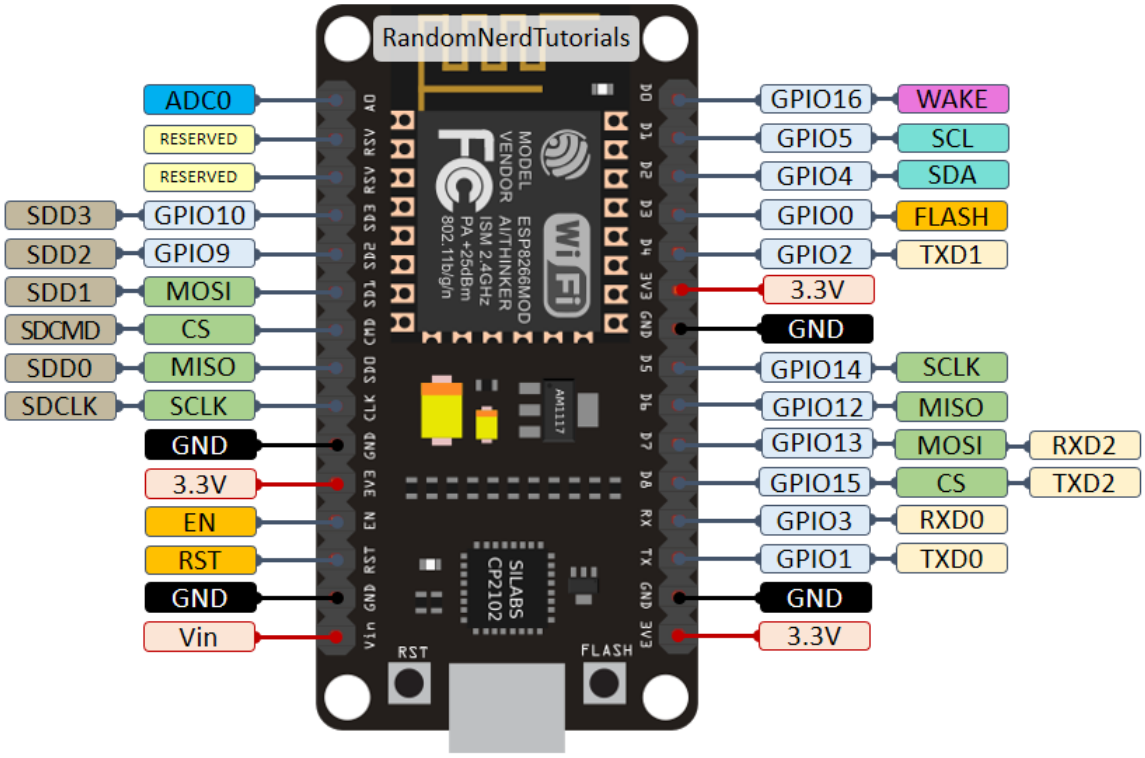
  - UART,
  - OTA.

- **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

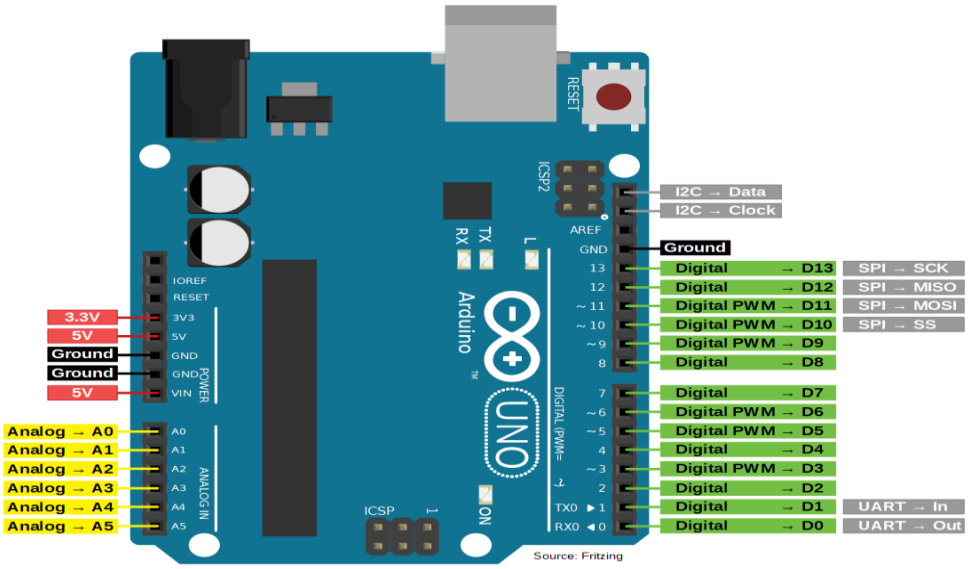


Rysunek 2. ESP8266 PinOut

Arduino Uno

- 16 Mhz CPU
- 32 KiB pamięci flash
- 2 KiB SRAM
- 1 KiB EEPROM
- Ilośc pinów I/O: 22

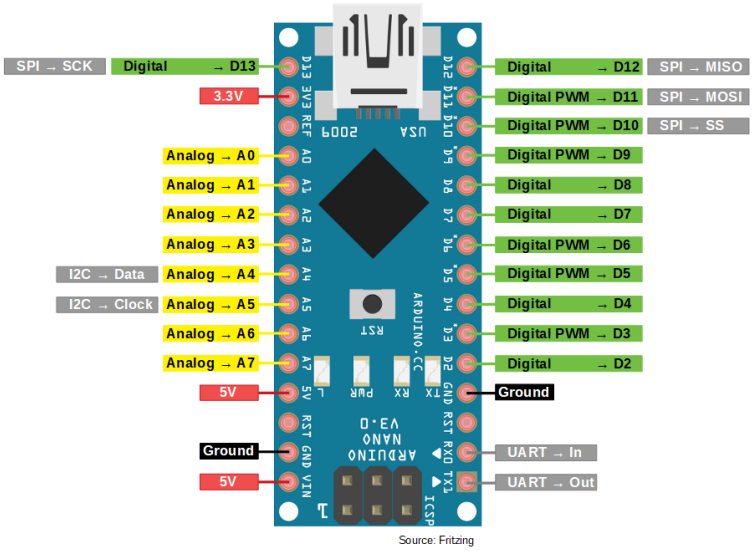
- Zasilanie: 7-12V
- Cena: 92.00 zł



Rysunek 3. Arduino Uno PinOut

Arduino Nano

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

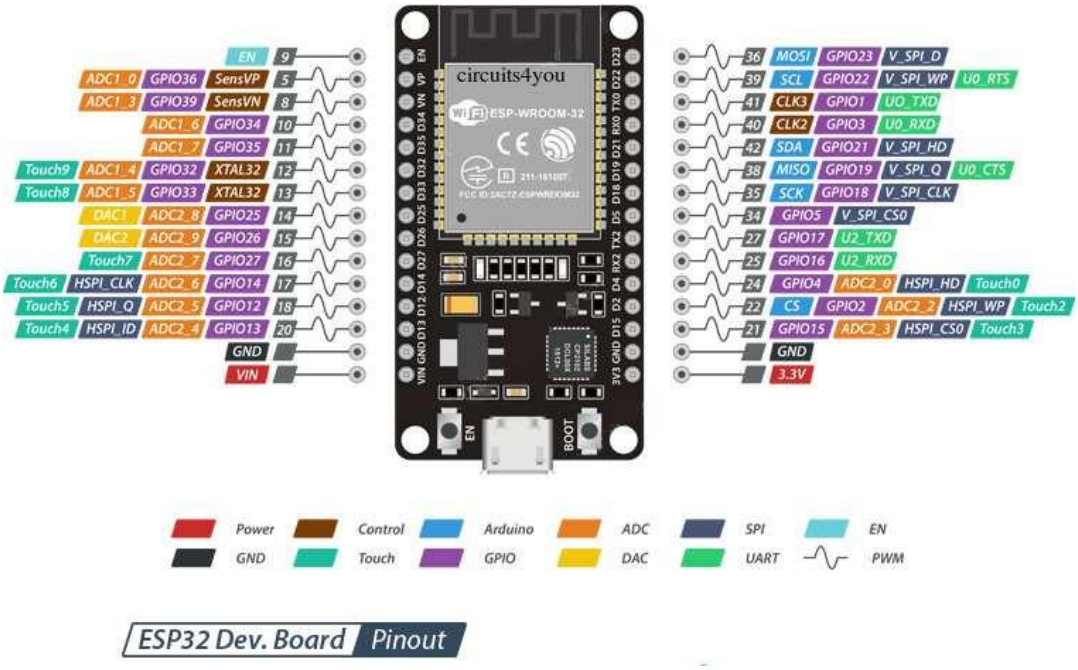


Rysunek 4. Arduino Nano PinOut

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



Rysunek 5. ESP32 PinOut

6.3. Pula czujników

DHT11

- **Ogólne**:

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

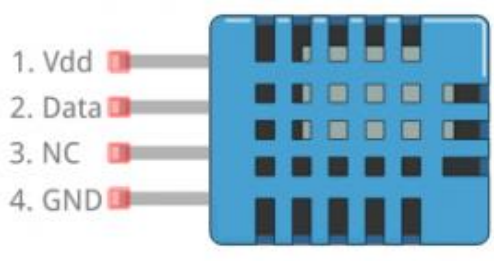
- **Wbudowany termometr**

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

- **Czujnik wilgotności**:

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

- **Cena: 4.33 zł**



Rysunek 6. DHT11 PinOut

DHT22 (AM2302)

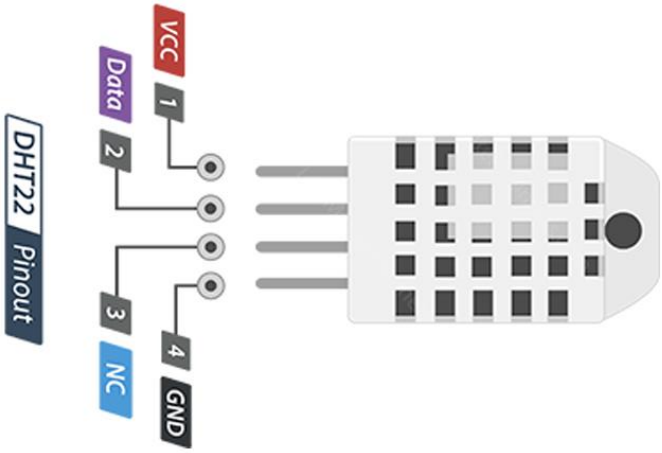
- **Napięcie zasilania: od 3,3 V do 6 V**
- **Średni pobór prądu: 0,2 mA**
- **Temperatura**

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

- **Wilgotność**:

  - Zakres pomiarowy: 0 - 100 % RH
  - Rozdzielczość: 8-bitów (±0,1 % RH)
  - Dokładność ±2 %RH*
  - Czas odpowiedzi: średnio 2 s

- **Cena: 24.90 zł**



Rysunek 7. DHT22 PinOut

DHT21 (AM2301)

- **Model: DHT21 / AM2301**
- **Temperatura**

  - Zakres pomiarowy: od -40 do +80 °C
  - Rozdzielczość: 0,1 °C
  - Dokładność: +/- 0,2 °C
  - Czas odpowiedzi: 2 s

- **Wilgotność:**

  - Zakres pomiarowy: 0 - 100 %RH
  - Rozdzielczość: 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 8. DHT21 PinOut

6.4. 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**.

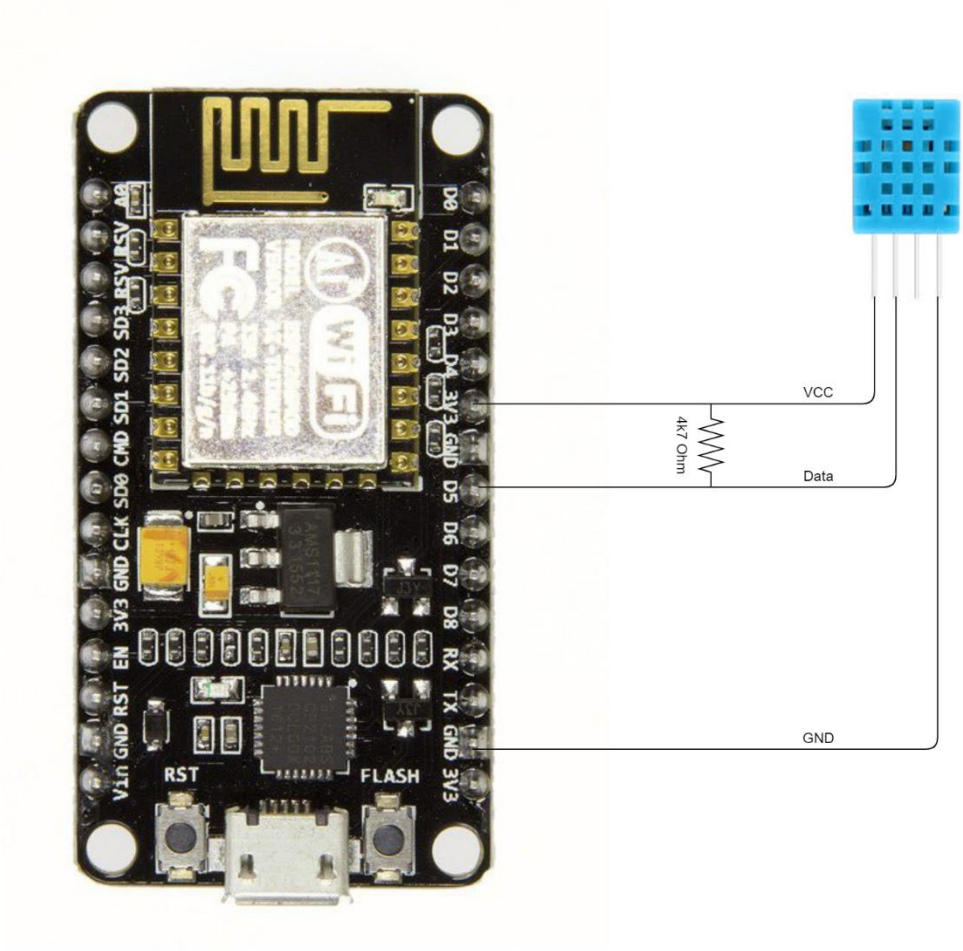
6.5. Wybór czujnika

Jedyne parametry które mogą rozróżniać 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.

6.6. Schemat elektryczny



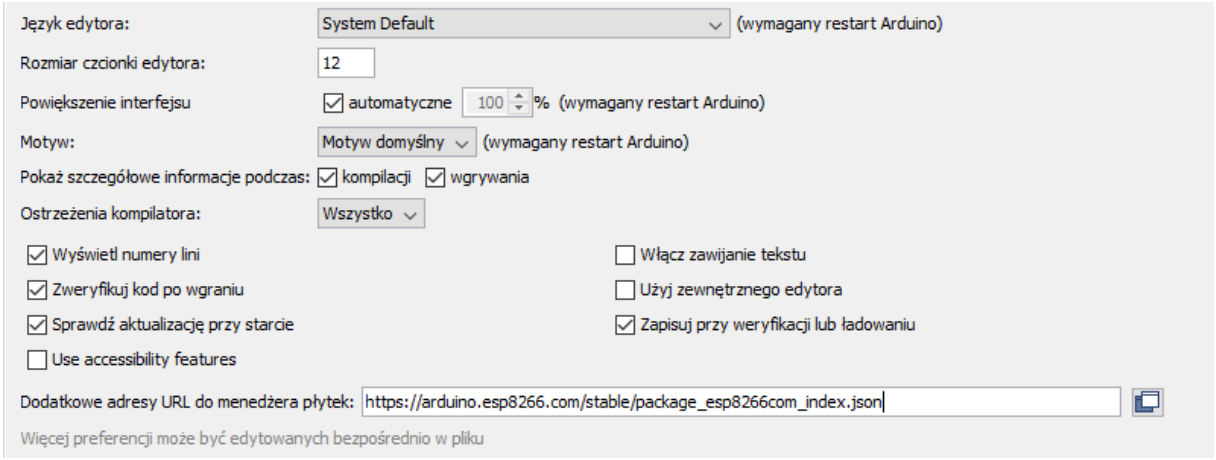
Rysunek 9. Schemat elektryczny

6.7. 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.

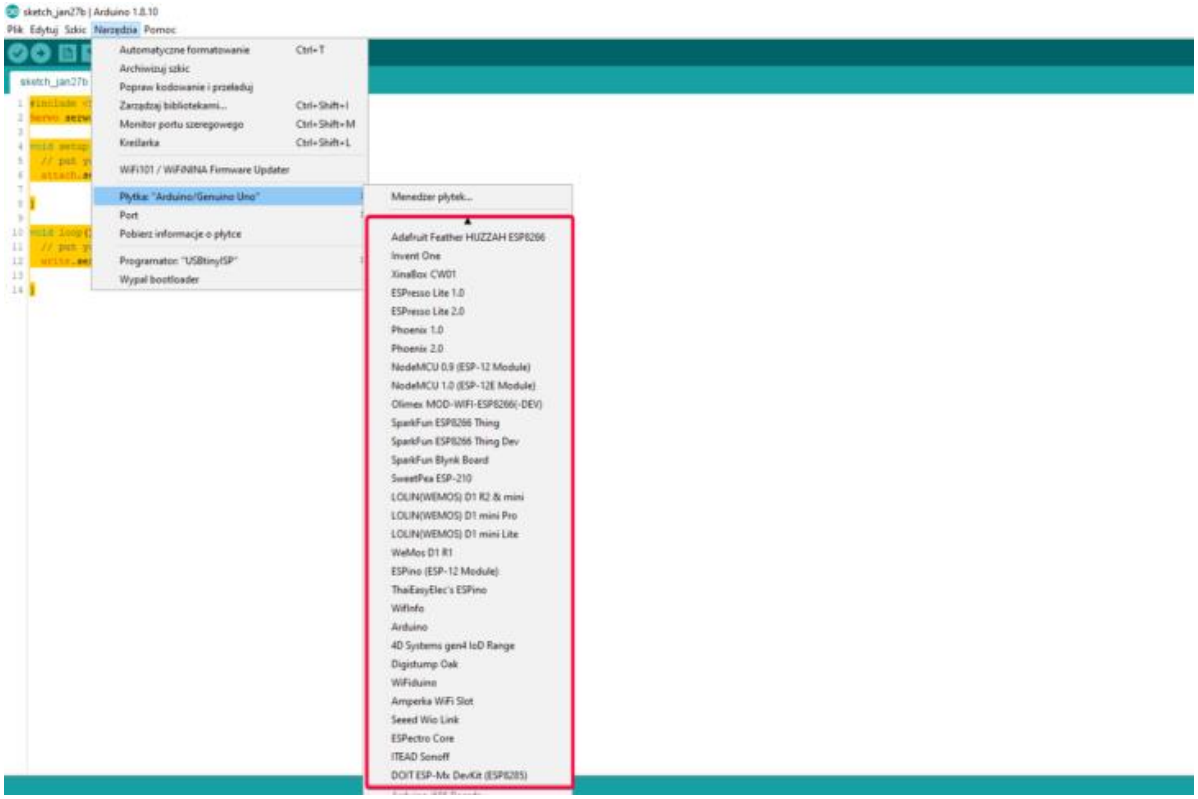
W Arduino IDE wybieramy opcję *Plik* > *Preferencję* i w polu *Dodatkowe adresy URL do menedżera płytek* wpisujemy poniższy adres:

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



Rysunek 10. Dodanie informacji o ESP8266 do Arduino IDE

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



Rysunek 11. Wybór płytek z ESP w Arduino IDE

6.8. 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

7. Wybrane warstwy OSI

Wykonanie: Katarzyna Czajkowska
Sprawdzenie: Arkadiusz Cichy

7.1. Model OSI

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

Warstwa aplikacji
Warstwa transportowa
Warstwa Internetu
Warstwa dostępu do sieci

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)

7.2. 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.

7.3. 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.

7.4. 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 liczbowa, 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.

7.5. 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.

8. Transmisja WiFi oraz TCP/IP

Wykonanie: Matusz Gurski
Sprawdzenie: Katarzyna Czajkowska

8.1. 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. Znacząco 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.**

8.2. 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 zmierzają. 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.

8.3. 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.

8.4. 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

8.5. 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.

9. 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 zapotrzebowań. 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.

9.1. 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.

9.2. Adresacja

W sieci, w której znajdować 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.

10. 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.

10.1. 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 konstrukturze.

```
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));
}
```

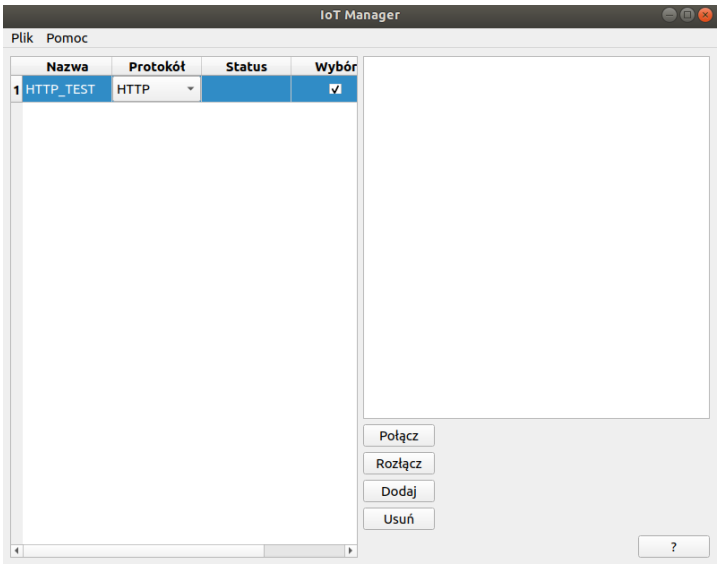
By odbierać dane z utworzonego modułu HTTP, wcześniej omówiony sygnał dataReady(data) musi zostać połączony ze slotem w aplikacji odpowiedzialnym za odiebranie odpowiedzi i jej dalsze przetwarzanie. W tym celu, w aplikacji utworzony został odpowiedni connect, jak widać łączy on sygnał dataReady(data) 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)));

...
```

10.2. 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.

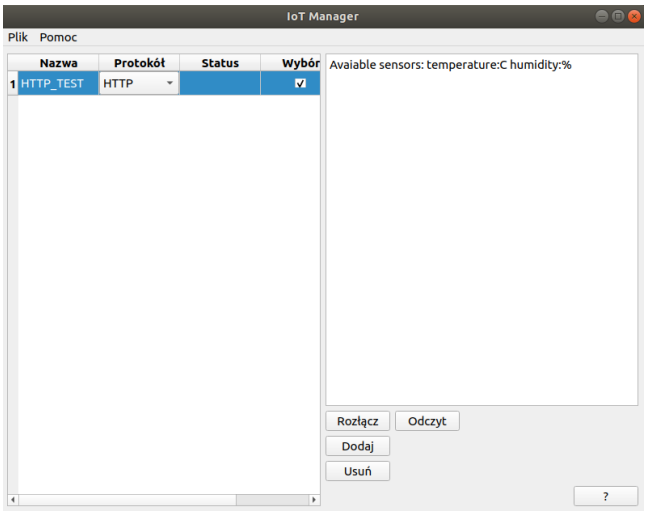


Rysunek 12. Wybór urządzenia

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.

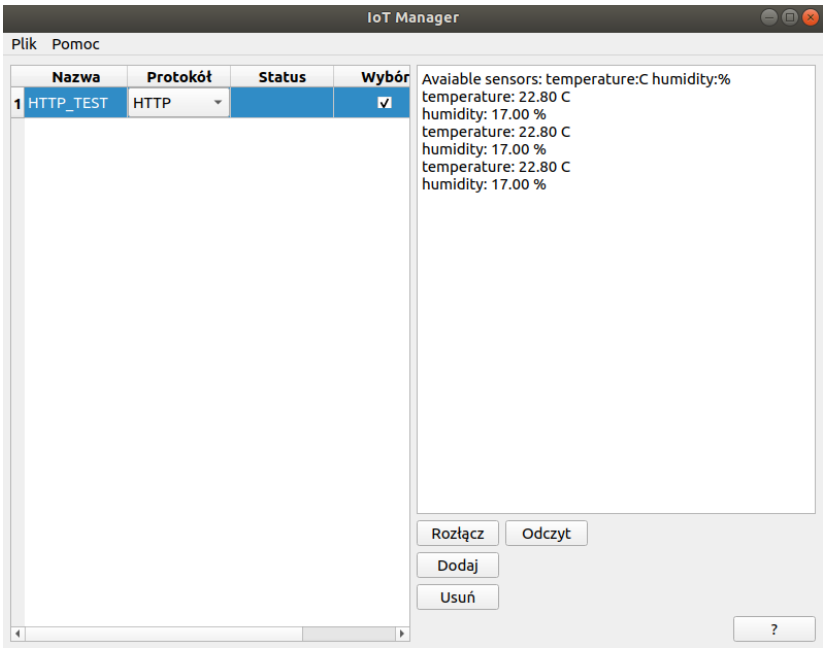
```
void UserInterface::on_pushButton_clicked()
{
    QModelIndexList selection = ui->iot_table->selectionModel()->selectedRows();
    if (selection.size() > 0){
        QModelIndex index = selection.at(0);
        int row = index.row();
        if (devices[row]->getProtocol() == 1){
            connected_device = 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 13. 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 14. Zakończenie połączenia

Przycisk odczyt sprawdza obsługiwany przez „połączone” urządzenie protokół, 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("Avaiable 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);
    }
}
```

11. Implementacja obsługi protokołu MQTT w aplikacji desktopowej

Wykonanie: Arkadiusz Cichy
Sprawdzenie: Szymon Cichy

11.1. 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.

11.2. Kluczowe funkcje biblioteki

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

```
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)d
        return nullptr;
    return subscription;
}
```

11.3. 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();
}
```

12. Program na platformę Android

Wykonał: Adam Krizar
Sprawdzenie: Katarzyna Czajkowska

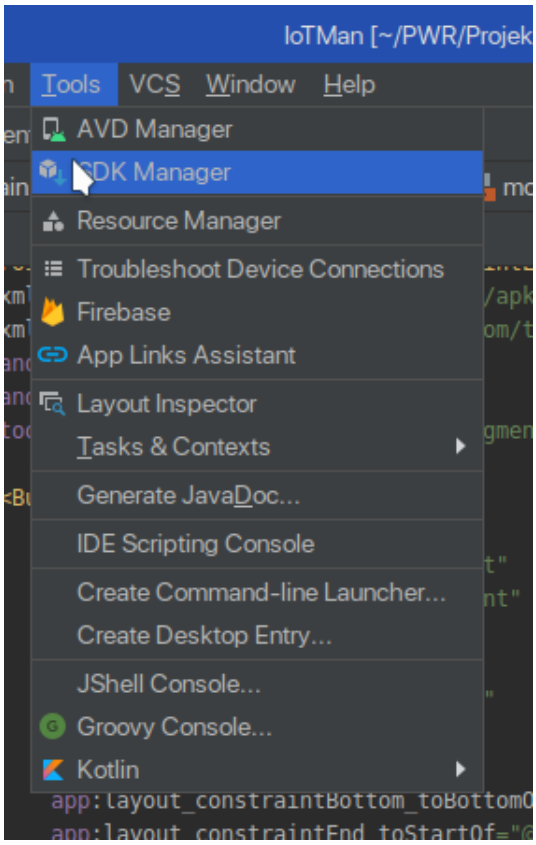
12.1. Środowisko Programistyczne

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.

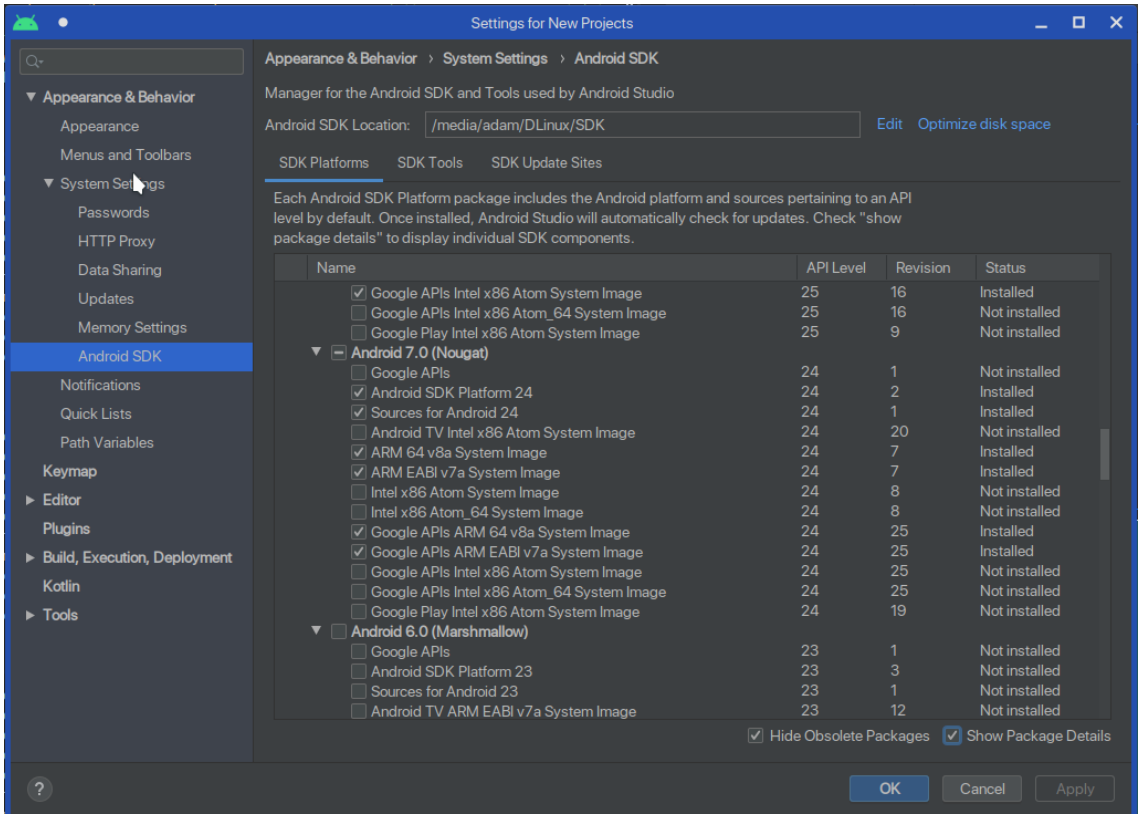
Rysunek 15. Informacje o Android Studio

Jako docelową wersje 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 z 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. Pobierze on automatycznie wszystkie elementy potrzebne użytkowania programu z wybraną SDK.



Rysunek 16. Dostęp do menedżera SDK

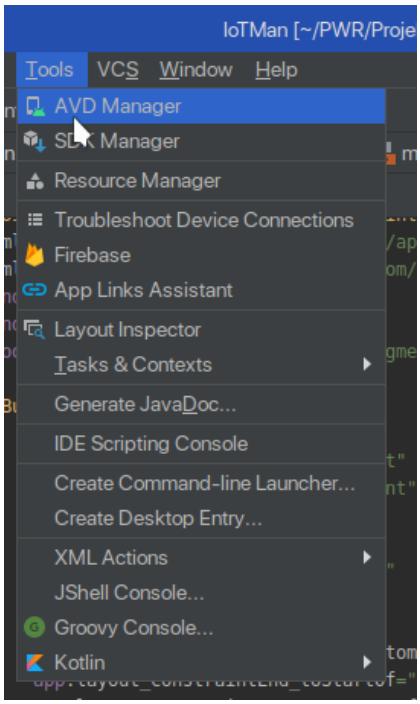


Rysunek 17. Wybór SDK

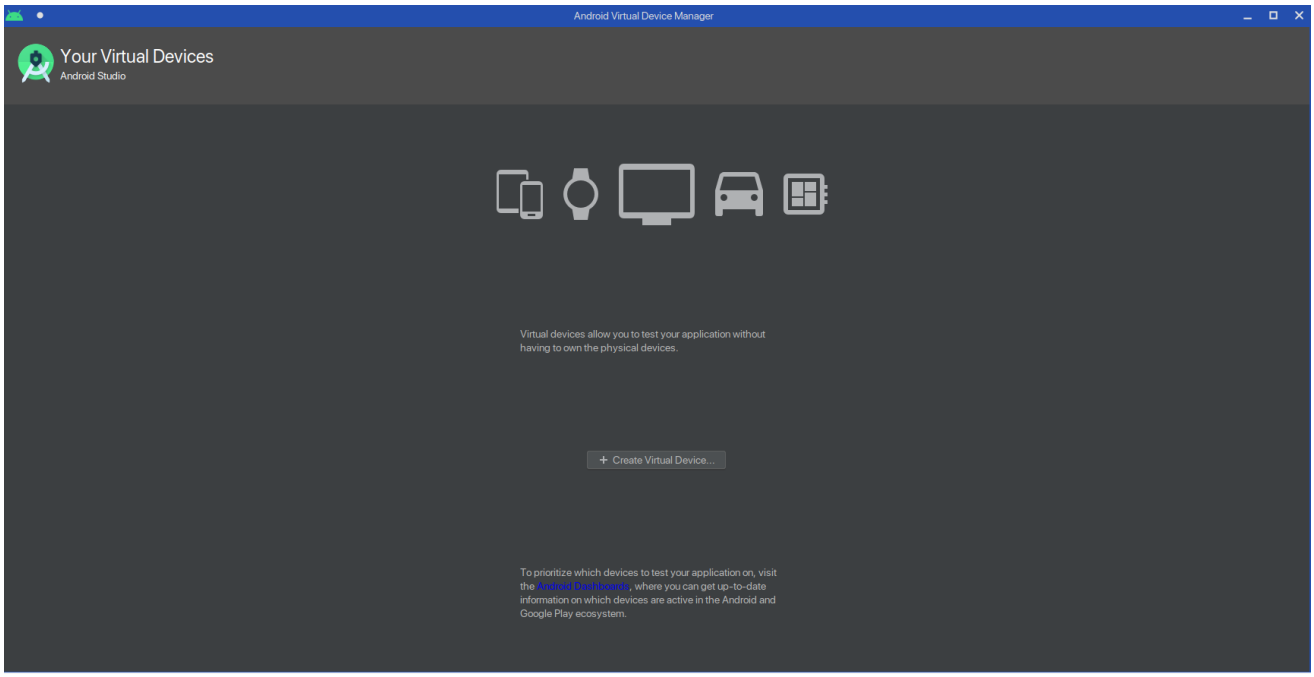
12.2. Uruchamianie aplikacji

Program może być przetestowany na istniejącym urządzeniu z systemem android w wersji co najmniej 7.0 (Nougat). 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)

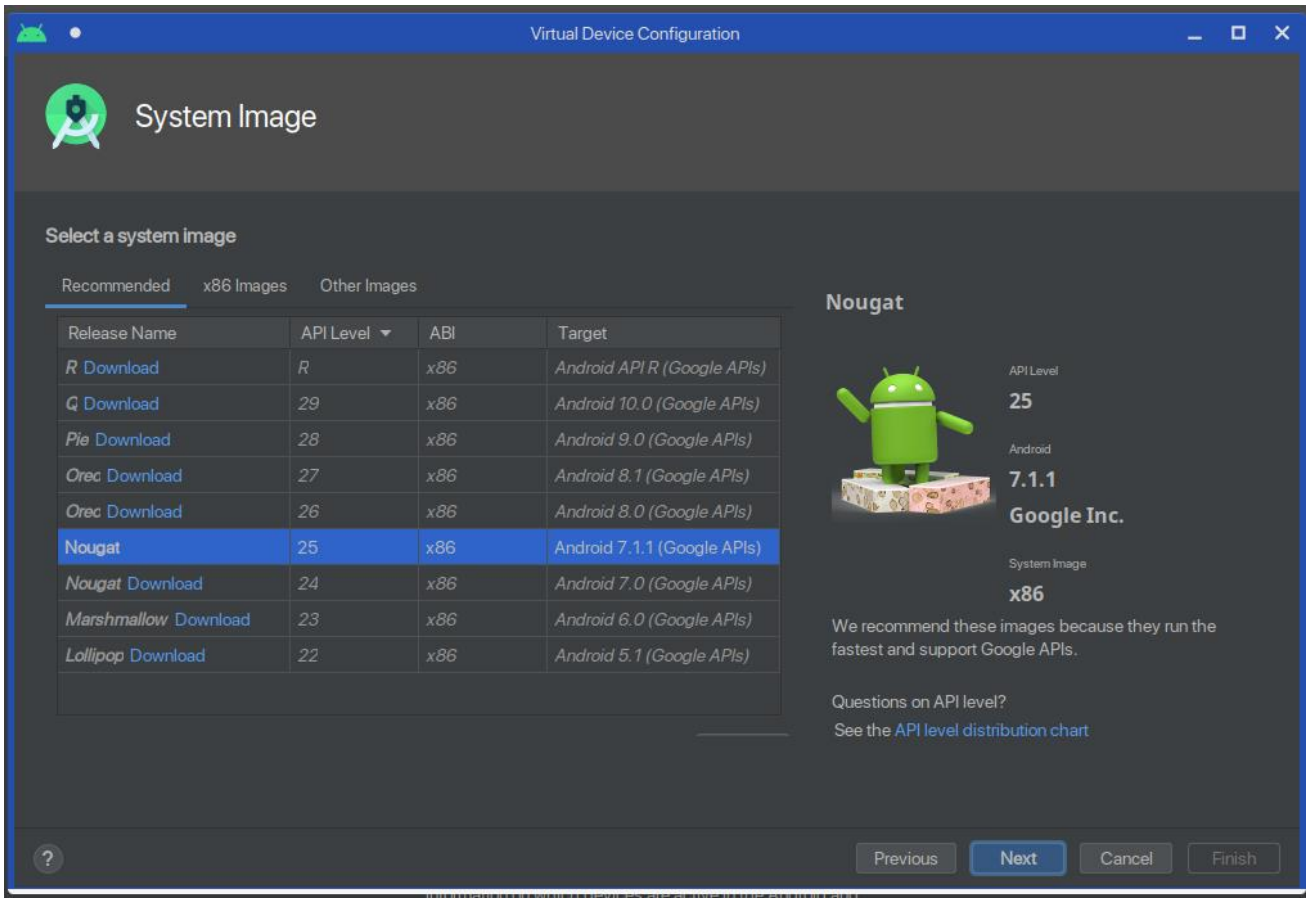


Rysunek 18. Lokalizacja menadżera AVD



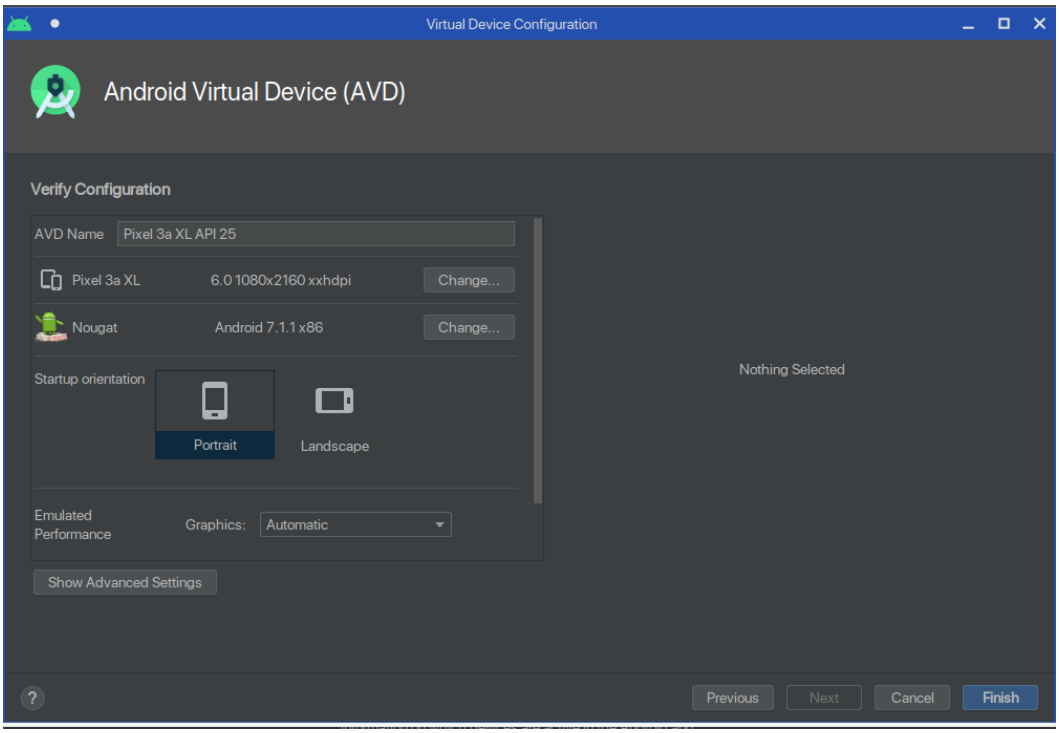
Rysunek 19. AVD menadżer

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 20.Wybór obrazu systemu wirtualnego urządzenia

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



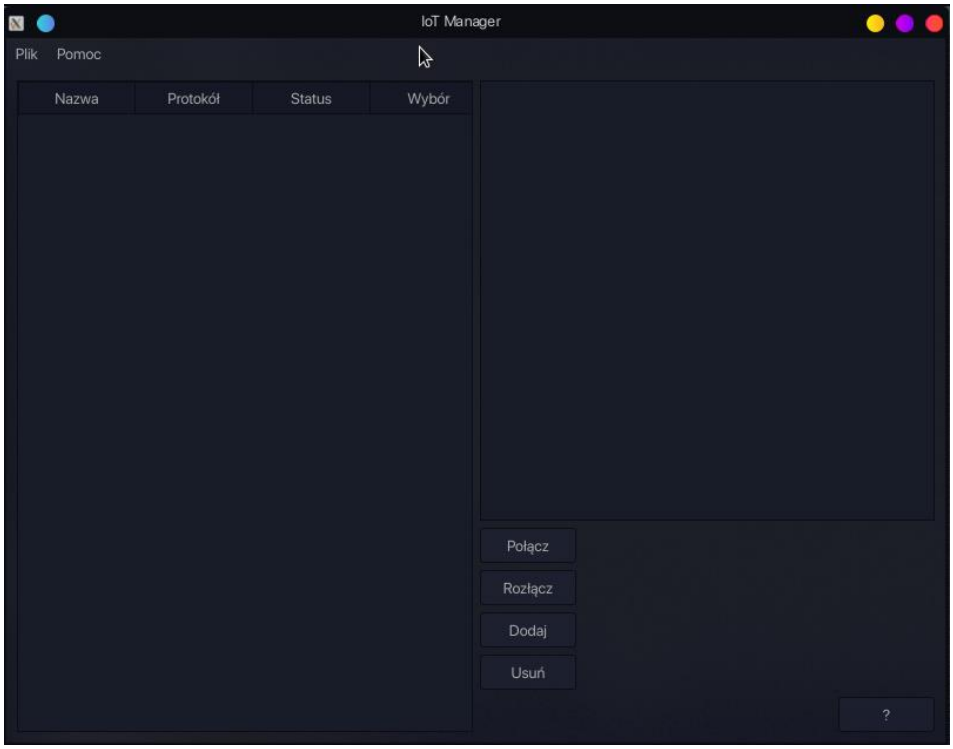
Rysunek 21. Końcowe ustawienia wirtualnego urządzenia.

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

13. Program na platformę Linux

Wykonanie: Katarzyna Czajkowska
Sprawdzenie: Mateusz Gurski

13.1. Podstawowe funkcje aplikacji desktopowej



Rysunek 22. Główne okno aplikacji desktopowej

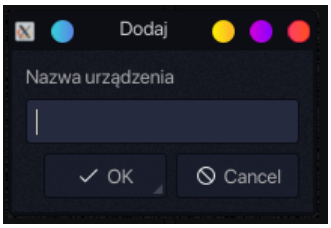
Dokładny wygląd okna zależny jest od dystrybucji Linuxa, na której program jest uruchomiony. Zrzut ekranu przedstawia program uruchomiony na Manjaro.

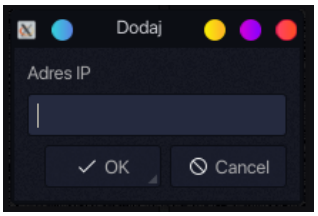
Jeżeli byłoby podłączone urządzenie, informacje na jego temat pojawiłyby się po lewej stronie okna, która służy za tabelę. Uruchomienie programu z przykładowym urządzeniem IoT pokazane jest w rozdziale 10.2, gdzie również pokazane jest działanie przycisków „**Połącz**” i „**Rozłącz**”.

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, ✗ oznacza, że urządzenie jest rozłączone;
- **Wybór** – odczyt z odhaczonych urządzeń będzie możliwy po wciśnięciu przycisku „Odczyt” (dostępnego dopiero po połączeniu)

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:

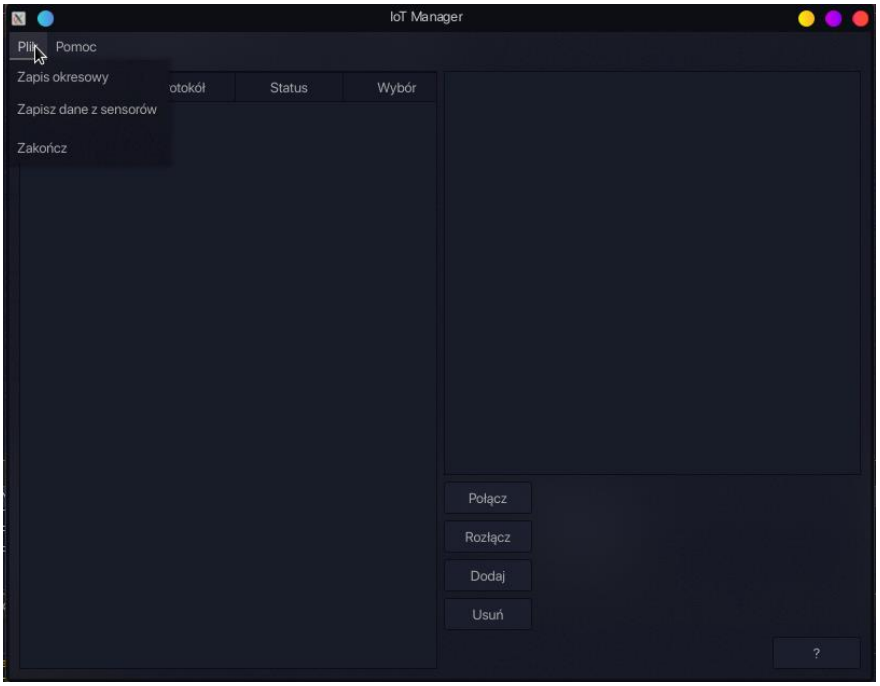




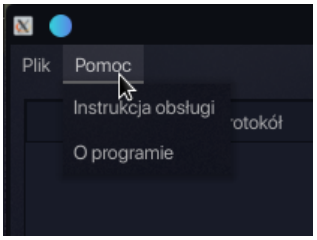
Przycisk „**Usuń**” służy do usuwania wybranego urządzenia z tabeli.

Przycisk „**?**” pozwalał będzie na wyświetlenie na bieżąco podpowiedzi na temat poszczególnych funkcji programu (po włączeniu go i kliknięciu/najechaniu myszą na przycisk lub inny element programu, do którego potrzebne mogą być wyjaśnienia)

13.2. Pasek menu

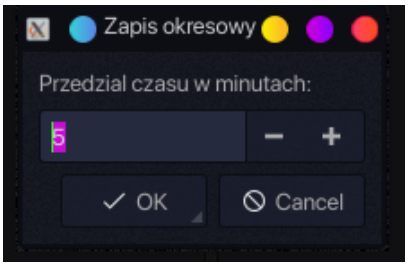


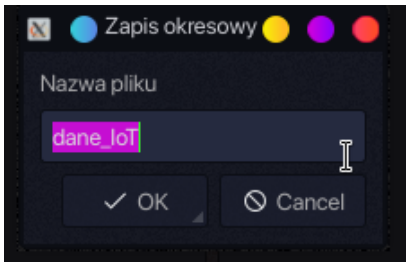
Rysunek 23. Wybór opcji „Plik” w menu



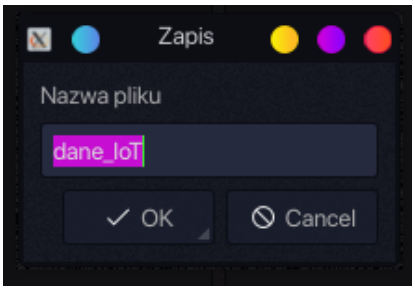
Rysunek 24. Wybór opcji „Pomoc” w menu

Całkowita użyteczność paska menu nie została jeszcze zaimplementowana, ale będzie on pozwalał na:
- **Zapis okresowy** – włączenie opcji zapisu okresowego do pliku – wyświetla okno dialogowe, w którym można wprowadzić przedział czasowy, co jaki dokonywany będzie zapis (1-60 minut) a następnie okno dialogowe pozwalające na wprowadzenie nazwy pliku, do którego dane mają być zapisywane.

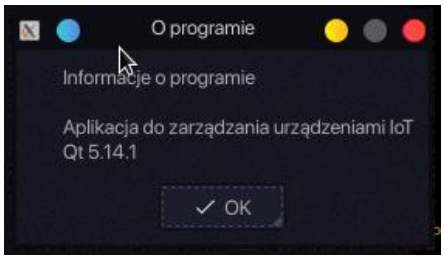




- **Zapisz dane z sensorów** – otwarcie okna dialogowego pozwalającego na wpisanie nazwy pliku, pod jaką chce się zapisać dane.



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



14. Oprogramowanie urządzenia IoT – http

Wykonanie: Mateusz Gurski
Sprawdzenie: Szymon Cichy

14.1. Instalacja bibliotek

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:

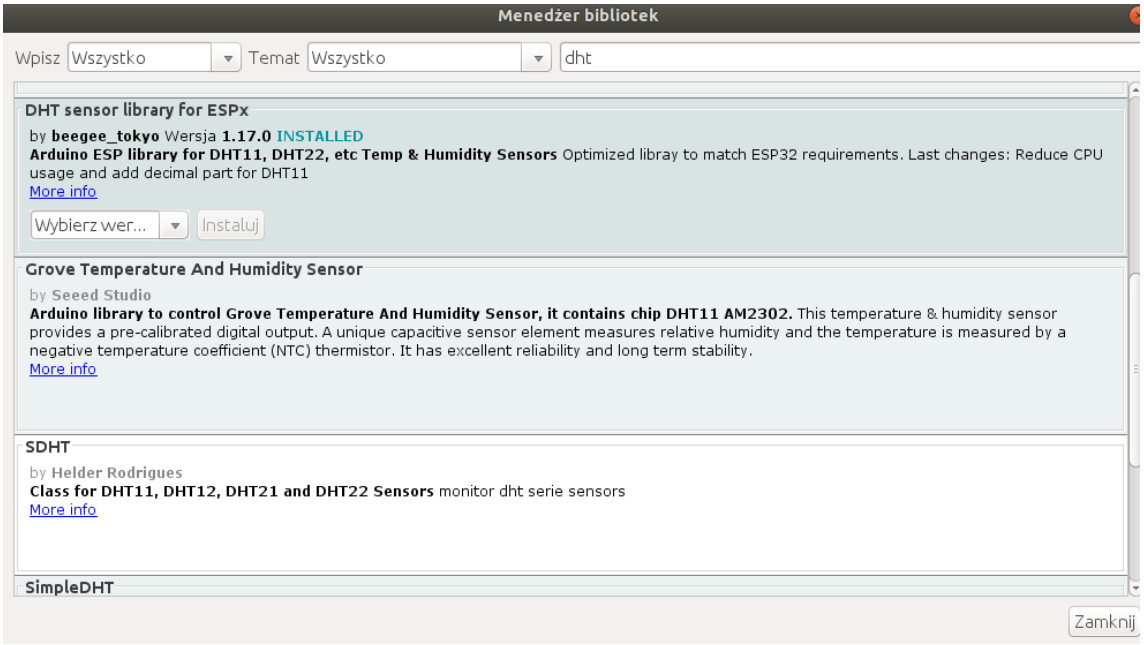
https://github.com/me-no-dev/ESPAsyncWebServer/archive/master.zip

oraz

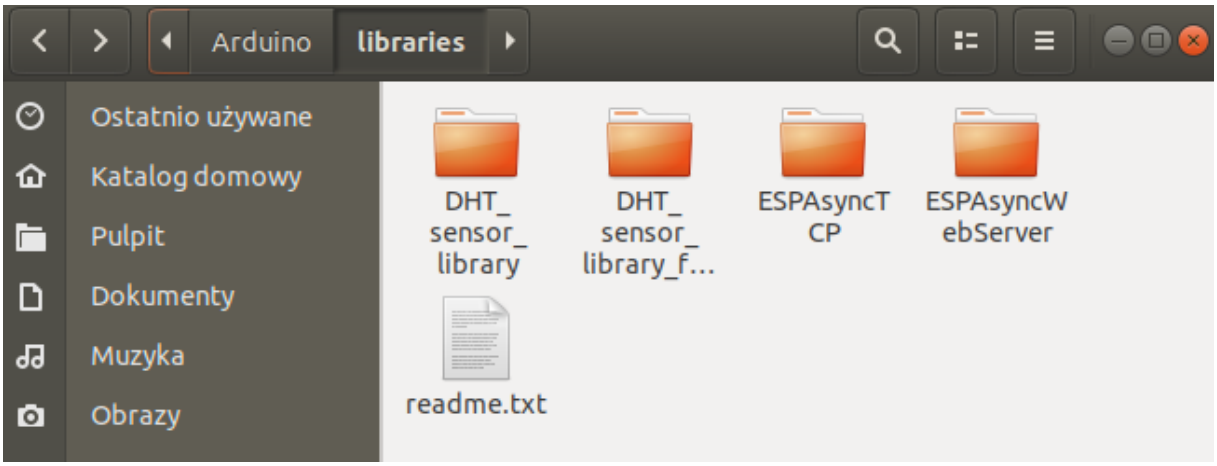
https://github.com/me-no-dev/ESPAsyncTCP/archive/master.zip

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 25. Menedżer bibliotek programu Arduino IDE



Rysunek 26. Katalog zawierający potrzebne biblioteki

14.2. Opis utworzonego oprogramowania

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;
```

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, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", "temperature:C humidity:%");
  });


  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *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());
  });
```

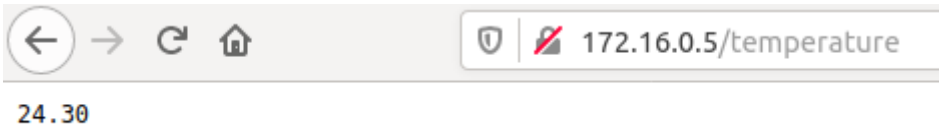
Inicjalizacja obiektu dht dla parametrów DHTPIN, w naszym przypadku jest to pin D5 oraz wersji czujnika DHT11. Potem, w następnej linii – uruchomienie serwera http.

```
  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 /temperature.



Rysunek 27. Odczyt temperatury

15. Kosztorys

Autor Adam Krizar

Głównym kosztem w realizacji naszego projektu są urządzenia IoT koniecznie 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

16. 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]**

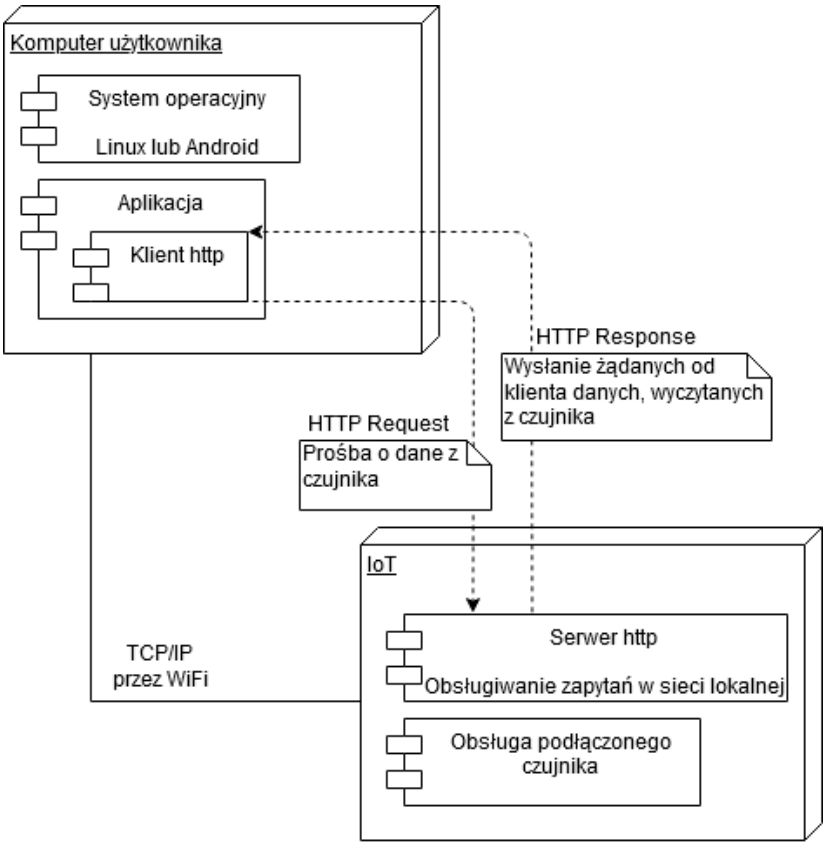
  Przeniesie aplikacji na system android. Przygotowanie drugiego urządzania IoT oraz przetestowanie protokołu MQTT.

- **Instalacja [07.05]**
- **Testy użytkownika [21.05]**

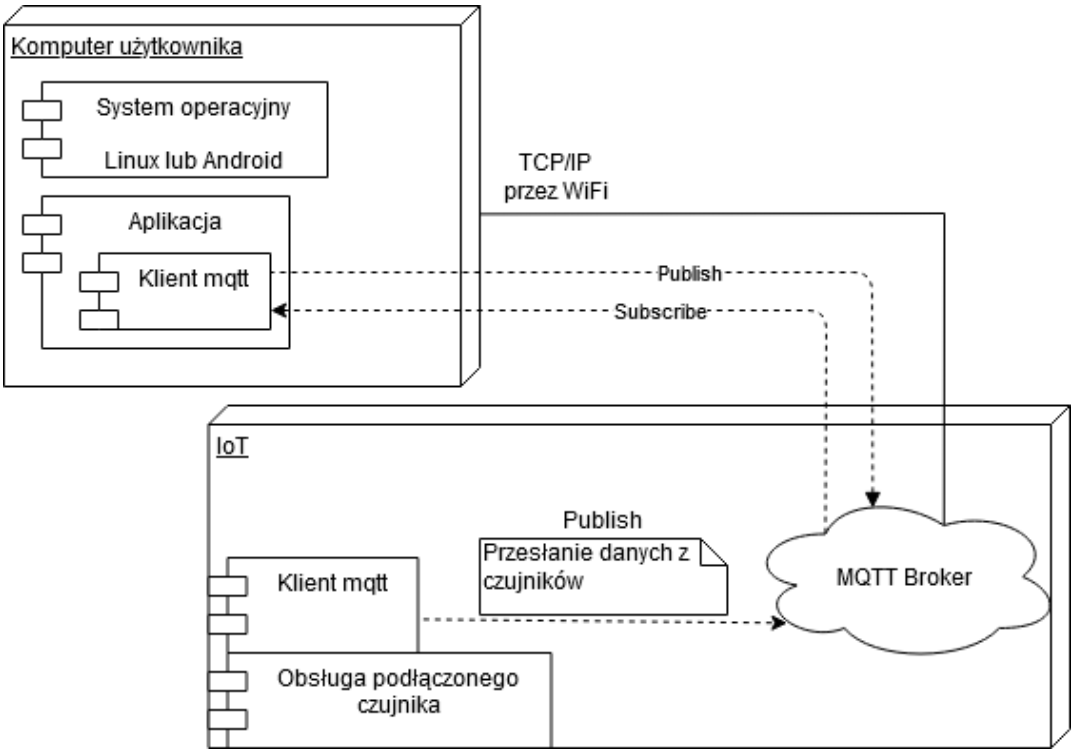
  Wprowadzenie ewentualnych korekt w projekcie interfejsu użytkownika zgodnie z uwagami użytkownika końcowego.

- **Oddanie projektu do użytku [04.06]**
- **Prezentacja naszych osiągnięć [10.06]**

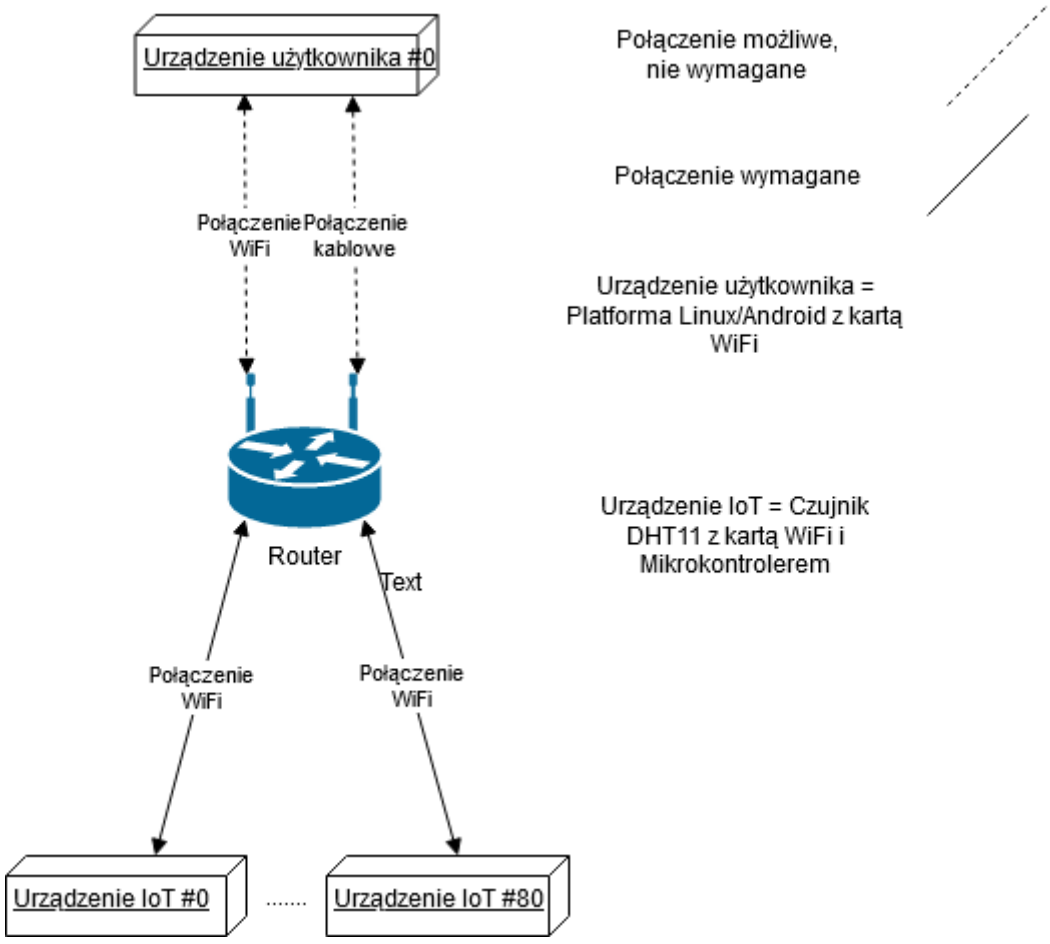
17. Propozycja rozwoju systemu



Rysunek 2.. Obsługa protokołu HTTP



Rysunek 3.. Obsługa protokołu MQTT



Rysunek 4.. Ogólna propozycja użycia aplikacji

18. 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