

|  |  |  |  |
| --- | --- | --- | --- |
| 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** | *9 kwiecień 2020* |

Spis treści

[1. Plan zadań 4](#_Toc37013321)

[ Grupa 1 5](#_Toc37013322)

[ Grupa 2 5](#_Toc37013323)

[ Grupa 3 5](#_Toc37013324)

[ Grupa 4 5](#_Toc37013325)

[2. Opis zadania 5](#_Toc37013326)

[3. Wymagania 6](#_Toc37013327)

[4. Założenia 6](#_Toc37013328)

[5. Środowisko 6](#_Toc37013329)

[5.1. Instalacyjne 6](#_Toc37013330)

[5.2. Programistyczne 7](#_Toc37013331)

[6. Wybrane urządzenia/czujniki 7](#_Toc37013332)

[6.1. Wstęp 7](#_Toc37013339)

[6.2. Pula kontrolerów 7](#_Toc37013340)

[6.3. Pula czujników 10](#_Toc37013341)

[6.4. Wybór kontrolera 12](#_Toc37013342)

[6.5. Wybór czujnika 12](#_Toc37013343)

[6.6. Sposób programowania 13](#_Toc37013344)

[6.7. Oficjalna dokumentacja 14](#_Toc37013345)

[7. Wybrane warstwy OSI (TODO: Katarzyna Czajkowska) 14](#_Toc37013346)

[8. Transmisja WiFi oraz TCP/IP (TODO: Mateusz Gurski) 18](#_Toc37013347)

[9. Podział na podsieci (TODO: Katarzyna Czajkowska) 18](#_Toc37013348)

[10. Biblioteka MQTT (TODO: Arkadiusz Cichy) 18](#_Toc37013349)

[11. Biblioteka HTTP (TODO: Mateusz Gurski) 18](#_Toc37013350)

[12. Program na platformę Android (TODO: Adam Krizar/Szymon Cichy) 19](#_Toc37013351)

[13. Program na platformę Linux (TODO: Katarzyna Czajkowska) 19](#_Toc37013352)

[14. Oprogramowanie urządzenia IoT (TODO: Mateusz Gurski) 19](#_Toc37013353)

[15. Kosztorys 19](#_Toc37013354)

[16. Plan realizacji 19](#_Toc37013355)

[16.1. Pierwszy punkt kontrolny [19.03] 19](#_Toc37013356)

[16.2. Drugi punkt kontrolny [02.04] 19](#_Toc37013357)

[16.3. Trzeci punkt kontrolny [23.04] 19](#_Toc37013358)

[16.4. Instalacja [07.05] 19](#_Toc37013359)

[16.5. Testy użytkownika [21.05] 19](#_Toc37013360)

[16.6. Oddanie projektu do użytku [04.06] 19](#_Toc37013361)

[16.7. Prezentacja naszych osiągnięć [10.06] 19](#_Toc37013362)

[17. Propozycja rozwoju systemu 19](#_Toc37013363)

[18. Źródła 21](#_Toc37013364)

# Plan zadań

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)

# Opis zadania

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.

# Wymagania

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).

# Założenia

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.

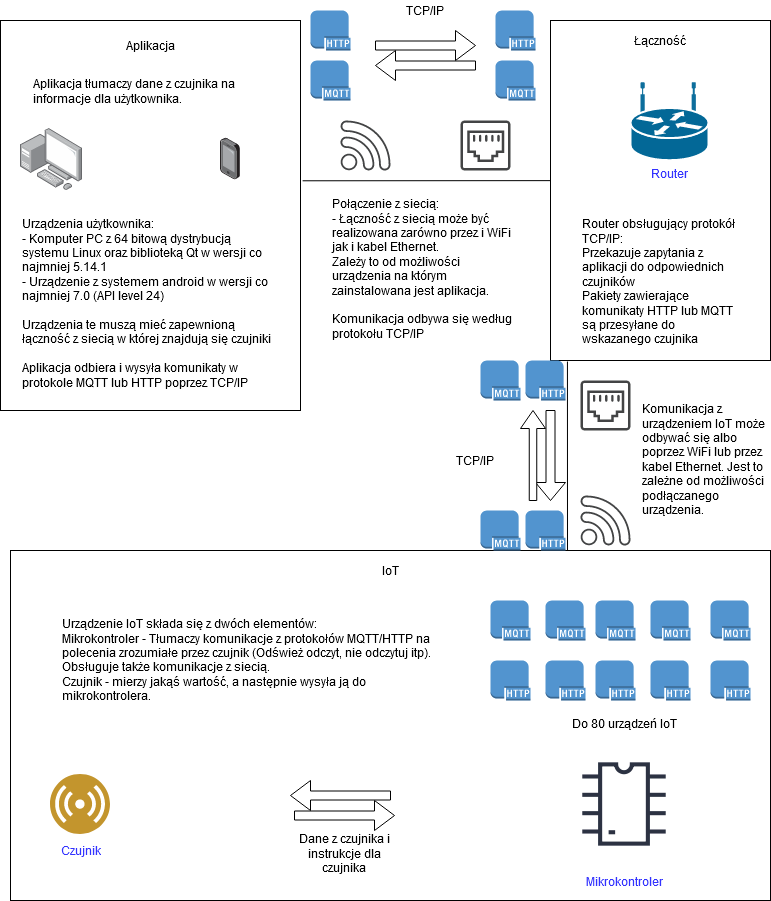
- Wymaganie implementacji minimum dwóch protokołów komunikacji z IoT: Implementacja protokołu HTTP oraz MQTT w naszej aplikacji oraz w testowym urządzeniu IoT. Te dwa protokoły zostały wyszczególnione jako przykładowe przez zgłaszającego oraz należą do najpopularniejszych rozwiązań na rynku co zapewni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

# Środowisko

## 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.

## 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.

# Wybrane urządzenia/czujniki



## 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.

## 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:

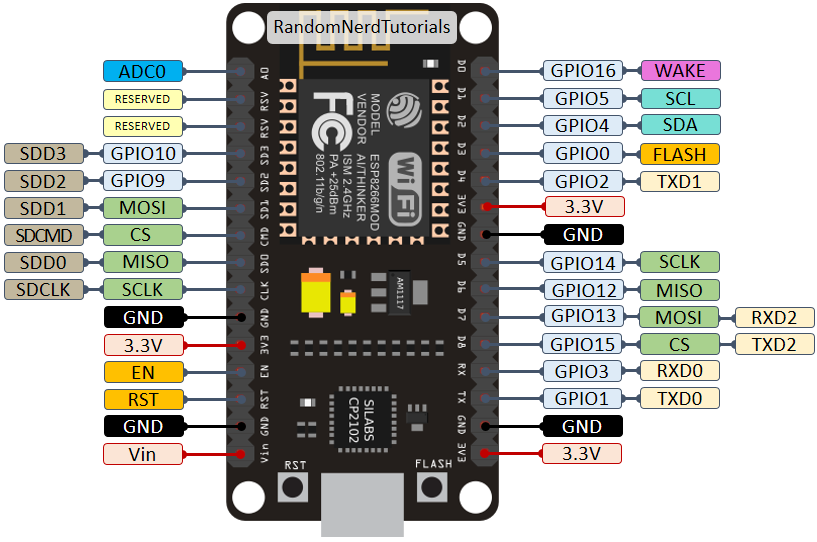
UART,  
OTA.

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

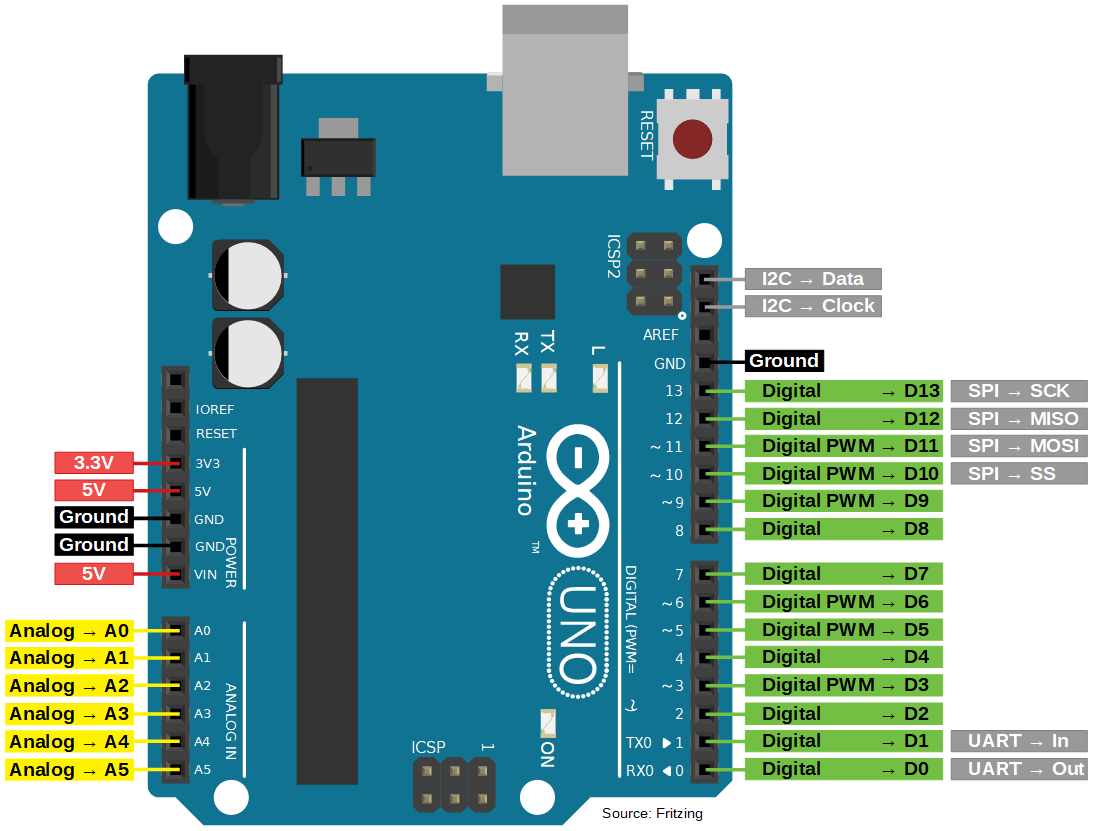
- 2 KiB SRAM

- 1 KiB EEPROM

- Ilośc pinów I/O: 22

- Zasilanie: 7-12V

Cena: 92.00 zł



Rysunek . Arduino Uno PinOut

**Arduino Nano**

- 16 Mhz CPU

- 32 KiB pamięci flash

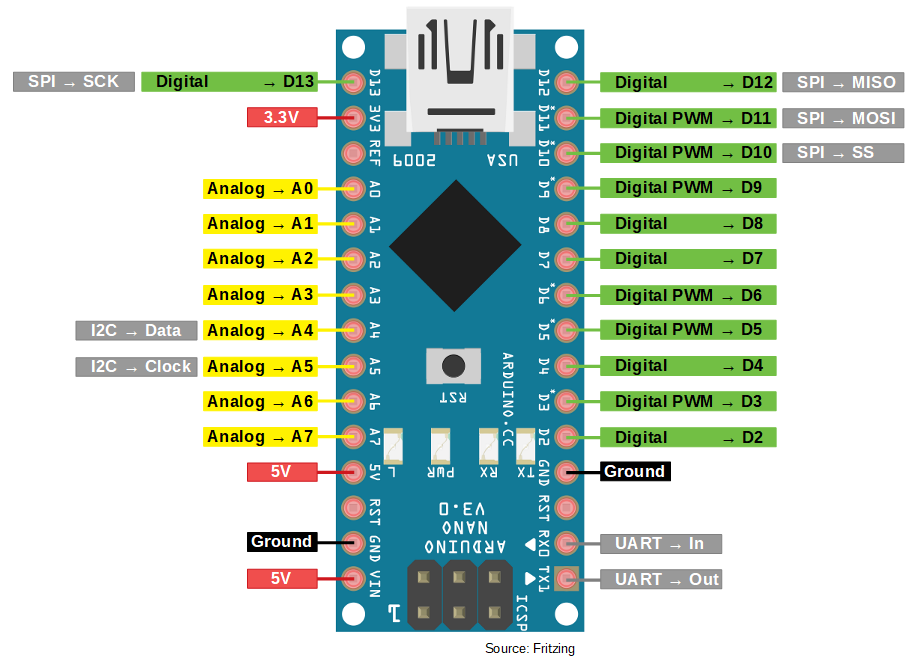
- 2 KiB SRAM

- 1 KiB EEPROM

- Ilośc pinów I/O: 14

- Zasilanie: 7-12V

Cena: 95.00 zł



Rysunek . 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 . ESP32 PinOut

## 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

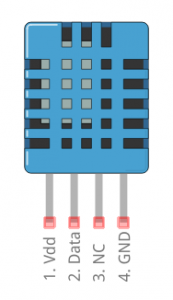
Dokładność: ±2°C

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:

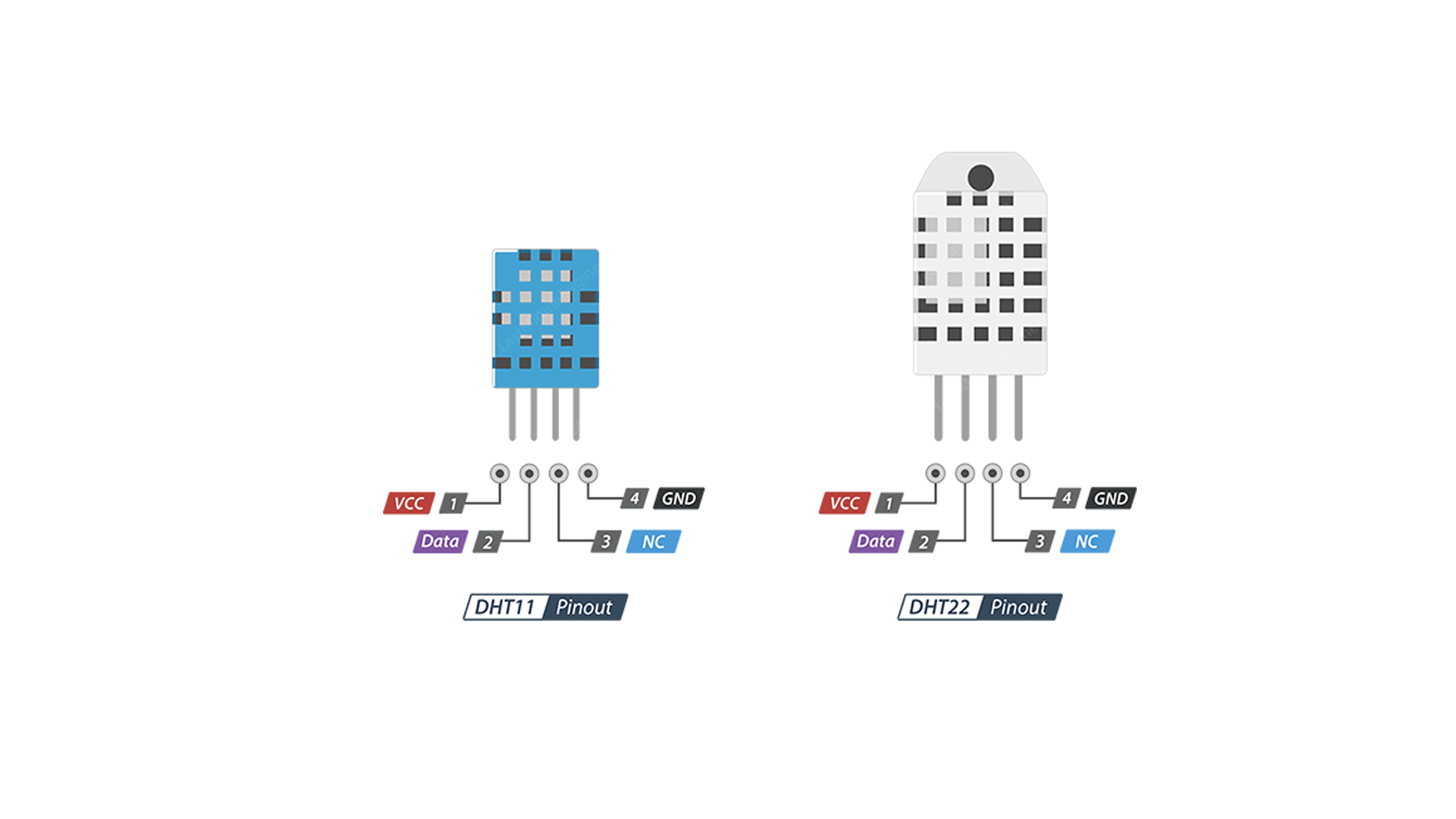
Zakres pomiarowy: 0 - 100 % RH

Rozdzielczość: 8-bitów (±0,1 % RH)

Dokładność ±2 %RH\*

Czas odpowiedzi: średnio 2 s

Cena: 24.90 zł



Rysunek . 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 . DHT21 PinOut

## 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**.

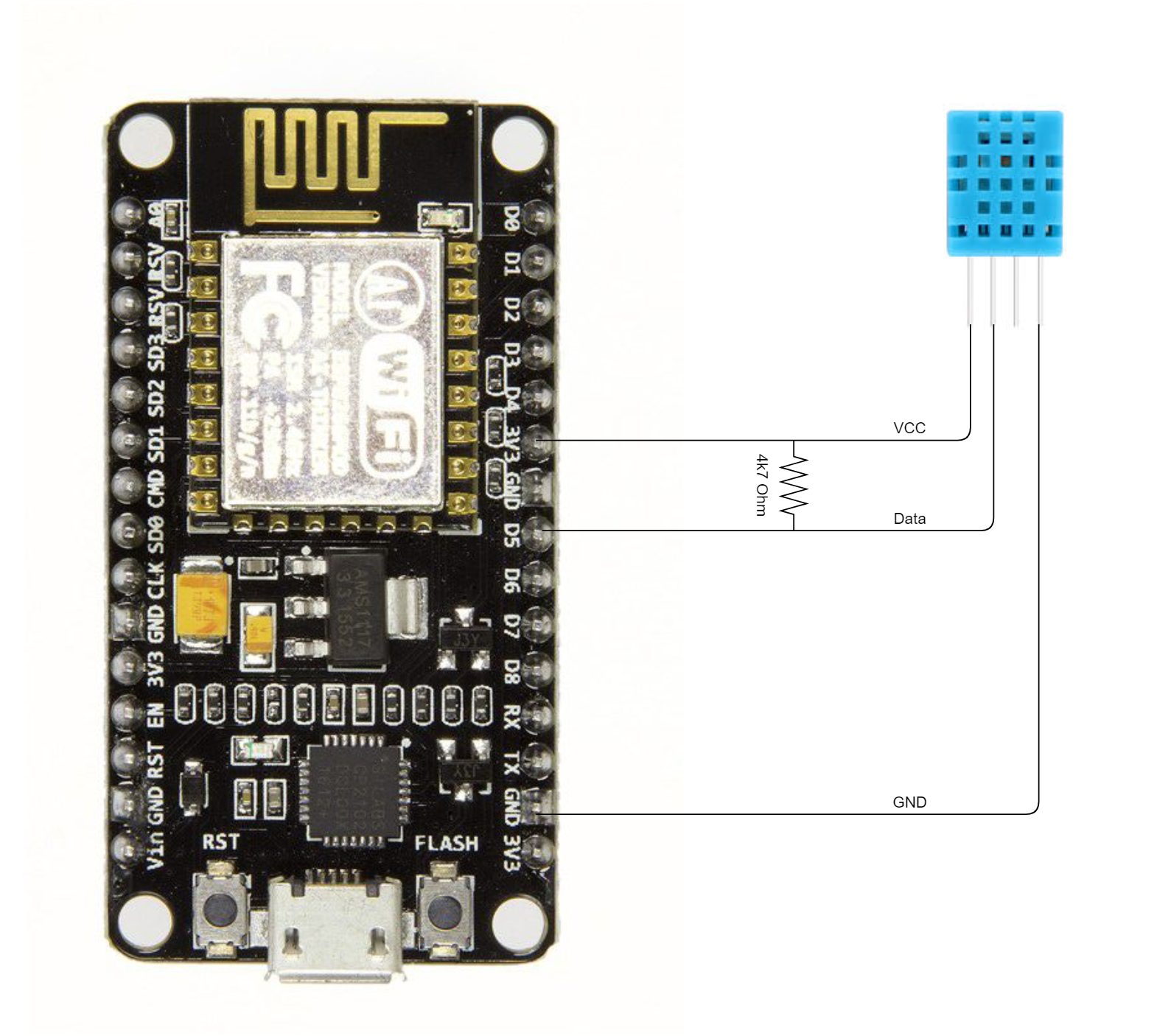
## Wybór czujnika

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.

Schemat elektryczny



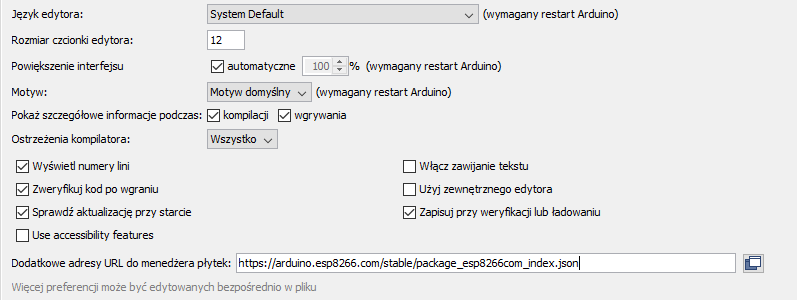
Rysunek . Schemat elektryczny

## 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.

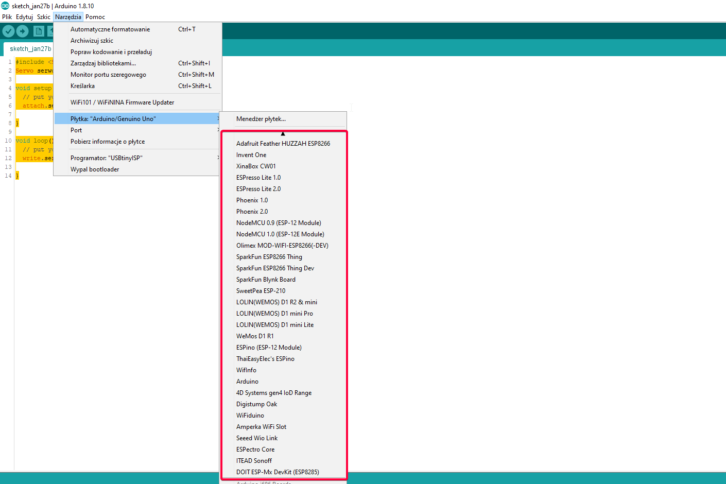
W Arduino IDE wybieramy opcję *Plik > Preferencję* i w polu *Dodatkowe adresy URL do menedżera płytek* wpisujemy poniższy adres:

https://arduino.esp8266.com/stable/package\_esp8266com\_index.json



Rysunek . 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 . Wybór płytek z ESP w Arduino IDE

## 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>

# Wybrane warstwy OSI (TODO: Katarzyna Czajkowska)

**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)

**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.

**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.

**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** – 01002 = 410, wskazuje wersję protokołu IP (dla IPv6 będzie to 01102 = 610)

- **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.

**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.

# Transmisja WiFi oraz TCP/IP (TODO: Mateusz Gurski)

# Podział na podsieci (TODO: Katarzyna Czajkowska)

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 2n-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 28-2 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 podsieci oraz 254-hostowych podsieci w tej samej sieci.

# Biblioteka MQTT (TODO: Arkadiusz Cichy)

# Biblioteka HTTP (TODO: Mateusz Gurski)

# Program na platformę Android (TODO: Adam Krizar/Szymon Cichy)

# Program na platformę Linux (TODO: Katarzyna Czajkowska)

# Oprogramowanie urządzenia IoT (TODO: Mateusz Gurski)

# 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

# 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]

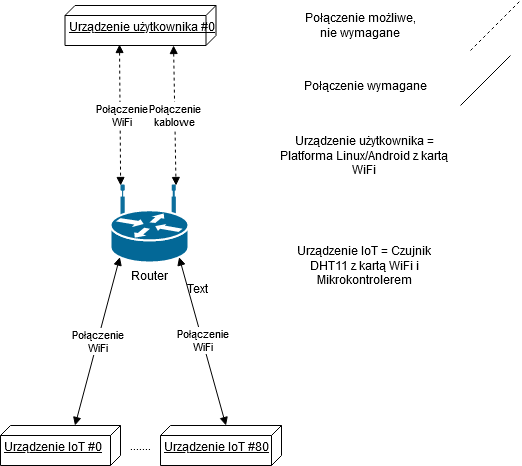
# Propozycja rozwoju systemu



Rysunek 2.. Obsługa protokołu HTTP



Rysunek 3.. Obsługa protokołu MQTT



Rysunek 4.. Ogólna propozycja użycia aplikacji

# 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>