Internet Rzeczy

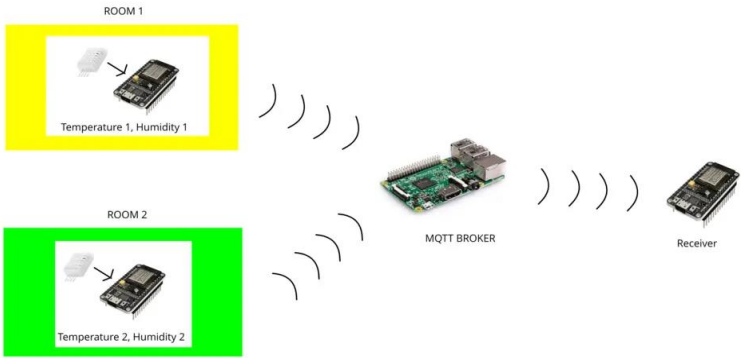
LAB 1-2 – Rafal Deska

Wprowadzenie

Internet rzeczy (IoT) rewolucjonizuje sposób, w jaki wchodzimy w interakcje z otaczającym nas światem. W czasie realizacji tego laboratorium użyjemy protokołu MQTT do komunikacji pomiędzy różnymi urządzeniami tworząc projekt urządzeń w architekturze cloud z wykorzystaniem mikrokomputerów z ESP8266 oraz platformy Raspberry PI. Urządzenia będą mierzyły proste parametry środowiska laboratoryjnego - temperaturę i wilgotność powietrza.W tym celu skonfigurujemy i zaprogramujemy ESP8266 do zbierania danych z czujników DHT11/DHT22 i przesyłania ich do Raspberry Pi działającego jako broker MQTT. Nauczymy się subskrybować i zarządzać wiadomościami MQTT odbieranymi przez Raspberry Pi oraz integrować dodatkowe funkcje z projektem IoT.

Czym jest MQTT?

MQTT (Message Queuing Telemetry Transport) to lekki i elastyczny protokół przesyłania wiadomości przeznaczony do komunikacji między urządzeniami z ograniczeniami przepustowości lub mocy obliczeniowej. Jest on szczególnie odpowiedni dla aplikacji IoT (Internet of Things) i M2M (Machine-to-Machine), w których kluczowe znaczenie ma oszczędność zasobów. MQTT opiera się na paradygmacie publikuj-subskrybuj, w którym urządzenia mogą publikować wiadomości w określonych tematach i subskrybować tematy zgodne ze swoimi potrzebami. Dzięki architekturze klient-serwer, MQTT obsługuje niezawodną i bezpieczną komunikację między urządzeniami o niewielkich zasobach, takimi jak czujniki i urządzenia EDGE, jest on szeroko stosowany w różnych sektorach, takich jak automatyka przemysłowa, automatyka domowa, a nawet sektor energetyczny. MQTT stał się podstawowym protokołem dla IoT z kilku powodów. Po pierwsze, jego lekkość pozwala urządzeniom o ograniczonych zasobach na wydajną komunikację. Co więcej, model MQTT umożliwia elastyczną i skalowalną komunikację między urządzeniami. W końcu, MQTT obsługuje niezawodne i bezpieczne połączenie między urządzeniami IoT, zapewniając bezpieczną transmisję danych. MQTT obsługuje QoS (Quality of Service), pozwalając zapewnić niezawodne dostarczanie danych, obsługuje również trwałość wiadomości nawet w przypadku tymczasowego rozłączenia urządzeń. Model publikuj/subskrybuj jest jedną z kluczowych cech MQTT. W tym modelu urządzenia są podzielone na dwie główne role: wydawców(publisher) i subskrybentów(subscriber). Wydawcy wysyłają wiadomości do określonego tematu, podczas gdy subskrybenci subskrybują temat, aby otrzymywać wiadomości związane z tym tematem. MQTT zarządza dystrybucją wiadomości do zainteresowanych subskrybentów, czyniąc komunikację wydajną i skalowalną. Do wdrożenia MQTT potrzebny jest broker MQTT, który działa jako pośrednik między urządzeniami. Broker odbiera wiadomości od wydawców i wysyła je do odpowiednich subskrybentów. Urządzenia IoT działają jako klienci MQTT i łączą się z brokerem.



Broker MQTT jest zatem sercem systemu MQTT. Działa jako pośrednik między urządzeniami wydawcy i subskrybenta, zarządzając dystrybucją wiadomości. Broker odbiera wiadomości od wydawców i wysyła je do subskrybentów na podstawie subskrybowanych tematów. Klienci MQTT to urządzenia, które łączą się z brokerem w celu wysyłania i odbierania wiadomości. Klienci mogą być zarówno wydawcami, jak i subskrybentami, w zależności od ich potrzeb.

BEZPIECZEŃSTWO MQTT

MQTT obsługuje uwierzytelnianie i autoryzację. Może używać mechanizmów takich jak nazwa użytkownika/hasło lub certyfikaty cyfrowe, obsługuje także bezpieczny transport danych poprzez TLS (Transport Layer Security).

W naszym projekcie użyjemy brokera MQTT Mosquitto zainstalowanego w systemie operacyjnym linux na mikrokomputerze raspberry pi podłączonym do laobratoryjnej sieci wifi. Mosquitto jest brokerem MQTT o otwartym kodzie źródłowym, opracowanym głównie przez Eclipse Foundation. Jest to jeden z najpopularniejszych brokerów MQTT i jest dostępny dla szerokiej gamy platform, w tym Linux, Windows i MacOS. Głównym plikiem konfiguracyjnym Mosquitto jest ***mosquitto.conf, w*** którym można określić ustawienia, takie jak port do nasłuchiwania, uprawnienia dostępu i inne opcje bezpieczeństwa.System zapewnia zestaw narzędzi wiersza poleceń do zarządzania i monitorowania brokera MQTT, takich jak mosquitto_pub do publikowania wiadomości i mosquitto_sub do subskrybowania tematów MQTT.

Opis systemu monitorowania środowiska IoT z protokołem komunikacyjnym MQTT

Elementy potrzebne do stworzenia układu

2x płytka prototypowa
1x czujniki DHT11/DHT22
1x rezystor 4,7 kΩ
1x karta SD o pojemności od 8 GB do 32 GB
1x wyświetlacz LCD 16×2
1x Raspberry PI
1x ESP-01

1x NodeMCU ESP8266
2x moduł zailający 3.3v
1x zasilacz do Raspberry Pi
przewody

Opcjonalne inne elementy w podstaci czujników i urządzeń wykonawczych niezbędne do realizacji zadań dodatkowych.

Do zaprogramowania układu ESP-01 konieczny będzie konwerter USB-RS232 o napięciu pracy 3,3V. Do instalacji systemu operacyjnego na Raspberry Pi konieczne będzie nagranie na kartę pamięci obrazu systemu ściągniętego ze strony www.raspberrypi.com

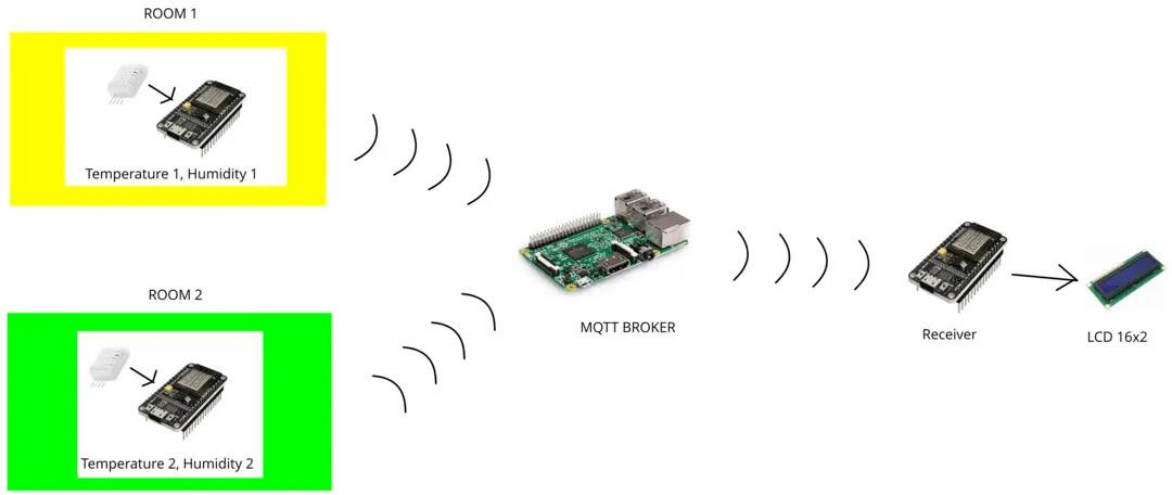
Projektowany i budowany przez nas system monitorowania parametrów środowiska połączy różne urządzenia i technologie w celu gromadzenia, przesyłania i wyświetlania w czasie rzeczywistym danych dotyczących temperatury i wilgotności otoczenia. System składa się z dwóch ESP8266 (jednego esp-01 i jednego bardziej zaawansowanego), czujników DHT11/DHT22, Raspberry Pi działającego jako broker MQTT oraz wyświetlacza LCD do wizualizacji danych.

W celu zaliczenia tego laboratorium w ciągu dwóch zajęć musisz zrealizować następujące zadania obowiązkowe (na ocenę dobrą)

1. **ESP8266 z DHT11/DHT22 (warstwa edge**):
    - ESP8266 (ESP-01) zostanie zaprogramowany do odczytu danych z czujnika DHT11/DHT22 w regularnych odstępach czasu.
    - Wykryte dane (temperatura i wilgotność) będą publikowane w dwóch różnych oddzielnych tematach MQTT, po jednym dla każdego typu.: ***temp _sensor_1***, ***hum_sensor_1***
2. **Raspberry Pi (Broker MQTT**):
    - Raspberry Pi będzie działał jako broker MQTT i odgrywał centralną rolę w systemie.
    - RPI będzie odbierał dane opublikowane przez ESP8266 (a po integracji z pracami innych studentów przez wszystkie czujniki) w odpowiednich tematach MQTT.
    - Przechowywał odebrane dane w tymczasowym buforze przed przesłaniem ich do drugiego ESP8266.
    - Utrzymywał stabilne połączenie z dwoma ESP8266, aby zapewnić ciągły odbiór danych.
3. **ESP8266 (wyświetlacz danych**):
    - ESP8266 zostanie zaprogramowany do subskrybowania tematów MQTT w celu odbierania danych o temperaturze i wilgotności z czujników.
    - Wykorzysta bibliotekę do sterowania wyświetlaczem LCD 16×2 w celu wyświetlania danych w czasie rzeczywistym.
    - Będzie aktualizował wyświetlacz o dane dotyczące temperatury i wilgotności za każdym razem, gdy otrzyma wiadomość MQTT z odpowiednich tematów.

Realizację zadań należy dokumentować na bieżąco opisując i dokumentując (również fotograficznie i poprzez zrzuty ekranu) kolejne etapy pracy. Wszystkie kody źródłowe należy zamieścić we własnym repozytorium GIT o nazwie MERITO_IOT_GDRD wraz ze stosownym plikiem .md zawierającym w minimalnej wersji imiona oraz numery indeksów wykonawców. Dodatkowo punktowana jest dokumentacja całego systemu.

W celu uzyskania wyższej oceny konieczna jest również realizacja zadań dodatkowych pojawiających się w szczegółowych opisach czynności do wykonania.



Rysunek 1Planowana architektura naszego systemu

1. Instalacja systemu Raspberry PI OS oraz połączenie z siecią WIFI

Zostanie zaprezentowane na zajęciach

2 .Instalacja brokera MQTT

zainstalujemy bardzo popularnego brokera MQTT o nazwie Mosquitto:

```
sudo apt-get update
sudo apt-get install mosquitto mosquitto-clients
```

a następnie włączamy i uruchamiamy usługę:

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

Domyślnie broker Mosquitto zezwala na połączenia anonimowe. Można go jednak skonfigurować tak, aby wymagał nazwy użytkownika i hasła, gdy klient łączy się z brokerem.

Ten akapit pokazuje, jak skonfigurować uwierzytelnianie za pomocą nazwy użytkownika i hasła dla brokera Mosquitto na Raspberry Pi. Na przykład użyjemy słowa ***student*** jako nazwy użytkownika i słowa ***merito*** jako hasła.

Możemy utworzyć plik hasła za pomocą narzędzia mosquitto_passwd.

Wymaga to uprawnień użytkownika root

```
sudo su
```

a następnie wpisujemy hasło superużytkownika

Po uzyskaniu uprawnień roota wpisz następujące polecenie:

```
mosquitto_passwd -c /etc/mosquitto/credentials student
```

Zostaniesz ***dwukrotnie*** poproszony o podanie hasła, które ma być powiązane z użytkownikiem student, wpisz ***merito.***

Polecenie utworzyło plik o nazwie ***credentials*** zawierający nazwę użytkownika (**student**) i zaszyfrowane hasło (**merito**) w folderze **`/etc/mosquitto/`**.

Następnie konieczna jest ręczna edycja pliku konfiguracyjnego.

```
nano /etc/mosquitto/mosquitto.conf
```

Należy dodać następujące polecenia:

```
listener 1883
allow_anonymous false
password_file /etc/mosquitto/credentials
```

```
systemctl restart mosquitto
```

Opuszczamy powłokę superużytkownika wpisując **`exit`** bądź wybierając kombinację **`CTRL-D`**

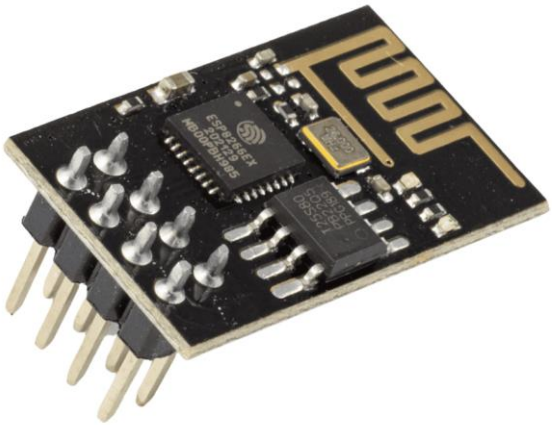
Aby zobaczyć wiadomości MQTT przesyłane z jednego z ESP8266 na Raspberry Pi, można użyć klienta ***mosquitto_sub*** MQTT dostarczonego z brokerem **Mosquitto.**

```
mosquitto_sub -h adres_ip_brokera -t hum_sens_1 -u mqtt_user -
P mqtt_password
```

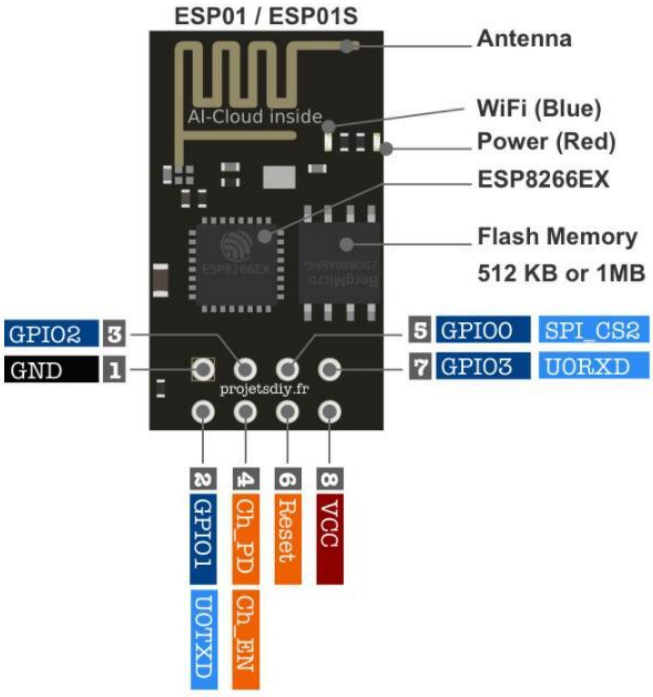
Po uruchomieniu polecenia powinieneś zobaczyć wiadomości MQTT przychodzące do terminala za każdym razem, gdy ESP8266 publikuje dane do określonego tematu. Wyświetlane komunikaty będą zawierać temperaturę i wilgotność wysyłane przez ESP8266.

Moduł ESP-01

ESP-01 to moduł WiFi do zastosowań w IoT z układem ESP8266 i pamięcią Flash 1MB. Antena PCB zapewnia łatwe podłączenie z siecią WiFi bez potrzeby podpinania zewnętrznych anten. Możliwa jest również komunikacja poprzez interfejs UART z wykorzystaniem komend AT co pozwala spiąć go z innymi mikrokontrolerami rozszerzając ich możliwości o komunikację WiFi. Układ posiada 2 wyprowadzenia GPIO.



Specyfikacja modułu
WiFi ESP8266 ESP-01
Pamięć Flash: 1 MB
Porty GPIO: 2
Napięcie zasilania: 3,3 V
Pobór prądu: do 300 mA
Kompatybilny ze standardem WiFi 802.11 b/g/n
Częstotliwość pracy: 2,4 GHz
Zabezpieczenie WiFi WPA/WPA2
Praca w trybie AP (Access Point), STA (Standalone) oraz AP+STA
Moc nadajnika: 19,5 dBm
Interfejs UART do komunikacji z mikrokontrolerem zewnętrznym lub programowania ESP8266 na module
Raster wyprowadzeń: 2,54 mm
Wbudowana antena PCB
Wymiary: 24,8 x 16 mm
Waga: 1,6 g



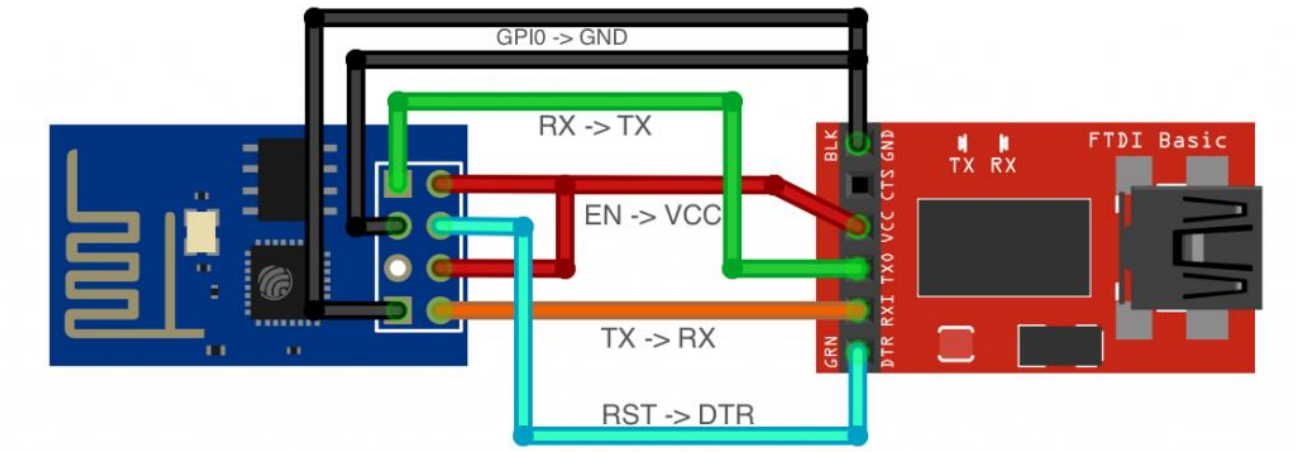
Konfiguracja środowiska Arduino IDE do pracy z ESP8266

1. Uruchomić środowisko Arduino IDE
2. Z menu File wybrać **Preferences** a następnie w polu **Additional boards manager URLs** wpisać: http://arduino.esp8266.com/stable/package_esp8266com_index.json
3. Zatwierdzić wpisany adres.
4. W boards manager wybrać płytkę ESP8266 i port pod którym widoczny jest konwerter USB-Serial.

Programowanie układu

UWAGA: Przed rozpoczęciem dokonywania połączeń upewnić się, że programator jest ustawiony na napięcie 3,3v

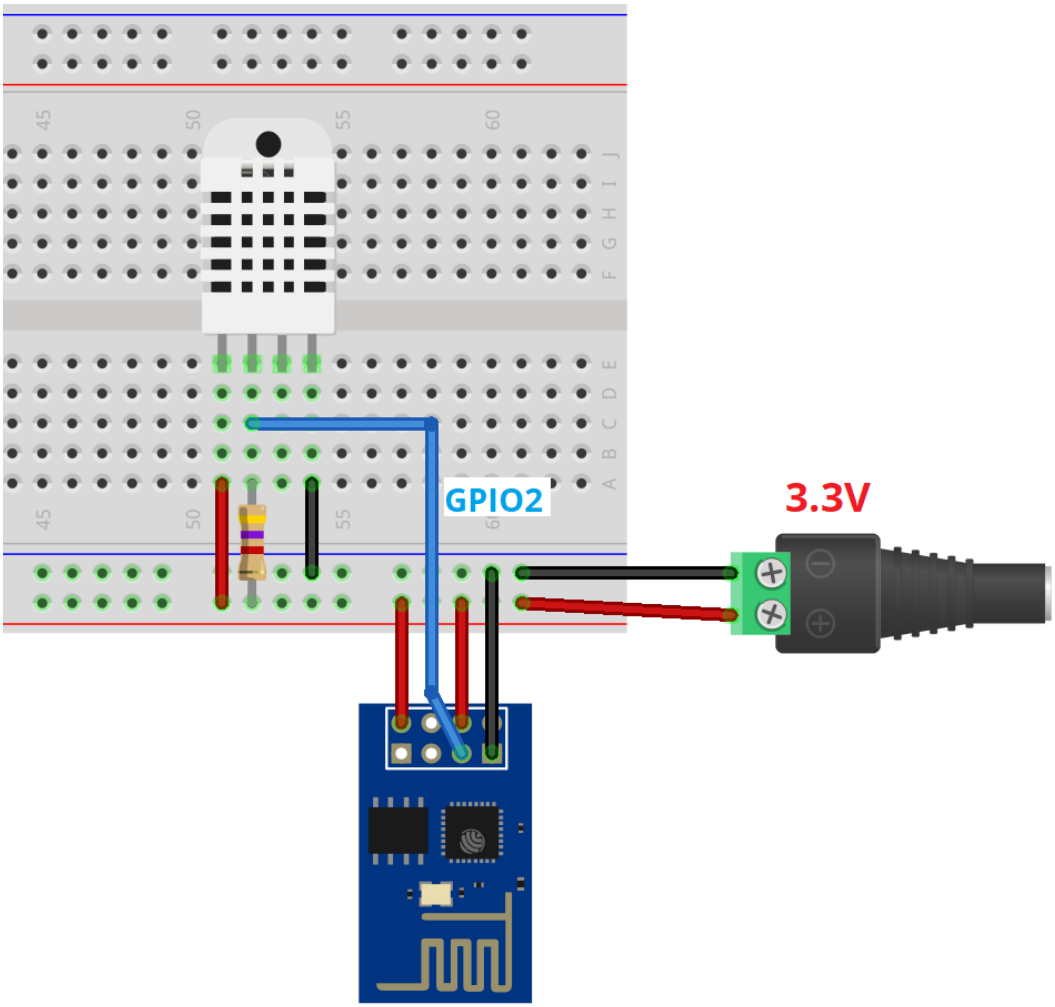
Z uwagi na minimalistyczny charakter płytki esp-01 konieczne jest specyficzne podejście do jej programowania. Schemat połączeń do programowania układu ESP-01 znajduje się poniżej:



W czasie programowania GPI0 musi być połączone z GND
W środowisku ARDUINO **IDE->Narzędzia** ustawiamy **Płytka: "Generic ESP8266 Module"** oraz port programatora przydzielony przez system Windows.
Najpierw sprawdzamy działanie układu programując go szkicem Blink
Odłączamy GPI0 od GND
Resetujemy moduł podłączając na moment EN do 3.3V
Jeśli wszystko poszło dobrze zaobserwujemy miganie diody podłączonej do pinu 2 modułu ESP-01. Przy powyższym połączeniu możliwe jest debugowanie wykonywania programu za pośrednictwem portu szeregowego.

Kolejnym zadaniem jest połączenie się z siecią bezprzewodową – proszę do tego celu wykorzystać odpowiedni szkic znajdujący się w przykładach środowiska Arduino IDE.

Po skutecznym przetestowaniu programowania układu i połączenia z siecią należy przystąpić do budowy układu docelowego z czujnikiem esp-01. W tym celu proszę również zaprojektować łatwy sposób programowania układu w przypadku konieczności zmiany oprogramowana.




```
#include "DHT.h"
#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>

#define WIFI_SSID "nazwa sieci"
#define WIFI_PASSWORD "haslo"

#define MQTT_HOST IPAddress(192, 168, 1, XXX)
#define MQTT_PORT 1883
// tematy MQTT
#define MQTT_PUB_TEMP "esp/dht/temperature"
#define MQTT_PUB_HUM "esp/dht/humidity"
```

```
#define DHTPIN 14  //pin połączony z czujnikiem - uwaga
dokumentacja!

// jaki czujnik jest polaczony?
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

// zmienne do przechowywania odczytow
float temp;
float hum;

AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;

WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
Ticker wifiReconnectTimer;

unsigned long previousMillis = 0;   // Stores last time
temperature was published
const long interval = 10000;        // Interval at which to
publish sensor readings

void connectToWifi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void onWifiConnect(const WiFiEventStationModeGotIP& event) {
  Serial.println("Connected to Wi-Fi.");
  connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected&
event) {
  Serial.println("Disconnected from Wi-Fi.");
  mqttReconnectTimer.detach(); // ensure we don't reconnect to
MQTT while reconnecting to Wi-Fi
  wifiReconnectTimer.once(2, connectToWifi);
}

void connectToMqtt() {
  Serial.println("Connecting to MQTT...");
  mqttClient.connect();
}

void onMqttConnect(bool sessionPresent) {
  Serial.println("Connected to MQTT.");
```

```
  Serial.print("Session present: ");
  Serial.println(sessionPresent);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason)
{
  Serial.println("Disconnected from MQTT.");

  if (WiFi.isConnected()) {
    mqttReconnectTimer.once(2, connectToMqtt);
  }
}

/*void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
  Serial.println("Subscribe acknowledged.");
  Serial.print("  packetId: ");
  Serial.println(packetId);
  Serial.print("  qos: ");
  Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
  Serial.println("Unsubscribe acknowledged.");
  Serial.print("  packetId: ");
  Serial.println(packetId);
}*/

void onMqttPublish(uint16_t packetId) {
  Serial.print("Publish acknowledged.");
  Serial.print("  packetId: ");
  Serial.println(packetId);
}

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

  dht.begin();

  wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
  wifiDisconnectHandler =
WiFi.onStationModeDisconnected(onWifiDisconnect);

  mqttClient.onConnect(onMqttConnect);
  mqttClient.onDisconnect(onMqttDisconnect);
  //mqttClient.onSubscribe(onMqttSubscribe);
  //mqttClient.onUnsubscribe(onMqttUnsubscribe);
  mqttClient.onPublish(onMqttPublish);
  mqttClient.setServer(MQTT_HOST, MQTT_PORT);
  // If your broker requires authentication (username and
password), set them below
```

```
  //mqttClient.setCredentials("REPlACE_WITH_YOUR_USER",
"REPLACE_WITH_YOUR_PASSWORD");

  connectToWifi();
}

void loop() {
  unsigned long currentMillis = millis();
  // Every X number of seconds (interval = 10 seconds)
  // it publishes a new MQTT message
  if (currentMillis - previousMillis >= interval) {
    // Save the last time a new reading was published
    previousMillis = currentMillis;
    // New DHT sensor readings
    hum = dht.readHumidity();
    // Read temperature as Celsius (the default)
    temp = dht.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    //temp = dht.readTemperature(true);

    // Publish an MQTT message on topic esp/dht/temperature
    uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP,
1, true, String(temp).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId:
%i ", MQTT_PUB_TEMP, packetIdPub1);
    Serial.printf("Message: %.2f \n", temp);

    // Publish an MQTT message on topic esp/dht/humidity
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM,
1, true, String(hum).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId
%i: ", MQTT_PUB_HUM, packetIdPub2);
    Serial.printf("Message: %.2f \n", hum);
  }
}
```