Dokumentacja projektowa

Sieć czujników zanieczyszczenia powietrza z uwzględnieniem wpływu wilgotności na uzyskane pomiary

Dokumentacja projektu zrealizowanego w ramach przedmiotu „[IoT] Internet of Things”   
na 2 semestrze studiów magisterskich Elektroniki i Telekomunikacji   
Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie

**Uczestnicy:**

Piotr Wanat

Leszek Kupczyk

Matusz Mazurkiewicz

Konrad Gujda

Kamil Sokołowski

Marcin Kordas

Spis treści

[1. Streszczenie 3](#_Toc532291051)

[2. Sprzęt 3](#_Toc532291052)

[1. Czujnik temperatury i wilgotności DHT11 - moduł niebieski: 3](#_Toc532291053)

[2. SparkFun CCS811 - czujnik czystości powietrza I2C: 3](#_Toc532291054)

[3. Oprogramowanie 4](#_Toc532291055)

[1. Streszczenie 6](#_Toc532291056)

[2. MQTT Broker 6](#_Toc532291057)

[a. Użyte biblioteki 6](#_Toc532291058)

[b. Kompilacja i działanie Brokera 6](#_Toc532291059)

[3. Firebase Manager 7](#_Toc532291060)

[a. Użyte biblioteki 7](#_Toc532291061)

[b. Kompilacja i działanie Firebase Manager 7](#_Toc532291062)

[c. Schemat 8](#_Toc532291063)

[4. Podsumowanie 8](#_Toc532291064)

[5. Wizualizacja 9](#_Toc532291065)

[a. Użyte biblioteki 9](#_Toc532291066)

[b. Parsowanie i działanie 9](#_Toc532291067)

[6. Podsumowanie 10](#_Toc532291068)

Dodać:

Schemat ideowy systemu

Odnośniki do źródeł

Jak dokładnie odpalić każdy z komponentów

Kod źródłowy jako link do GitHuba

Przeformatować dokumentację sensorów podobnie do dwóch ostatnich

Lessons learned section (jak udało się rozwiązać jakiś problem, to warto by to było opisane)

Wygenerowanie sztucznych danych (historyczne dane w formie kart – ewentualnych)

Dokumentacja zestawu sensorów

# Streszczenie

Zestaw sensorów składa się z czujników DHT22 (temperatura i wilgotność) oraz CCS811 (stężenie CO2 i zanieczyszczenie powietrza). Sensory podłączone są do płytki Wemos D1 mini z układem ESP8266 umożliwiającym komunikację za pośrednictwem WiFi. Całość została zaimplementowana z wykorzystaniem środowiska Arduino IDE.

# 2. Sprzęt

* Wemos D1 mini ESP8266 kompatybilny z Arduino
* CCS811
* DHT22

## 1. Czujnik temperatury i wilgotności DHT11 - moduł niebieski:

**Napięcie zasilania:** 3 V do 5,5 V. Do zasilenia zostanie użyte 3,3V.

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

**Czujnik wilgotności:** Zakres pomiarowy: 20 - 95%RH, Dokładność ±5%RH (RH-Relative Humidity)

**Protokół komunikacyjny:** Serial Interface (Single-Wire Two-Way)

**Schemat transmisji danych:**

Transmisja danych obejmuje 40bitów:

8bitów całkowite RH

+ 8bitów części dziesiętne RH

+ 8bitów całkowite T

+ 8bitów dziesiętne T

+ 8bitów sumy kontrolnej.

Dane będą zapisywane do zmiennych oraz transmitowane po każdym odczycie, każdy odczyt musi zostać wywołany przez hosta.

## 2. SparkFun CCS811 - czujnik czystości powietrza I2C:

**Napięcie zasilania:** 3,3V

**Czułość:**

Total Volatile Organic Compound (TVOC) od **0** do **1,187** ppb

eCO2 od **400** do **8,192** ppm

**Protokół komunikacyjny:** Komunikacja po I2C.

Wiele trybów pracy, w każdym inna częstotliwość pomiarów bez żądania hosta.

# 3. Oprogramowanie

Arduino IDE wraz bibliotekami CCS811, DHTesp (obsługa DHT22) oraz PubSubClient (implementacja klienta MQTT).

Komunikacja czujników z D1 mini odbywa się za pomocą magistrali I2C dla czujnika CCS811 oraz 1-Wire dla DHT22.

Poniższa linia kodu odpowiada za uaktywnienie portu D3 dla czujnika CCS811:

***CCS811 ccs811(D3,0x5B);***

Funkcja ***setup\_wifi()*** łączy ESP8266 do sieci przez Wi-Fi.

W tej funkcji używa się metody ***Wifi.begin()*** przyjmującej jako argumenty SSID sieci oraz hasło dostępowe. Następnie urządzenie oczekuje na połączenie przez Wi-Fi w pętli, która kończy się gdy sprawdzany parametr ***Wifi.status()*** informuje o udanym połączeniu.

Funkcja **reconnect()** działa w pętli, która kończy się w momencie, gdy nawiązane zostanie połączenie MQTT. Jej zadaniem jest ogłosić połączenie z serwerem funkcją ***client.publish()*** i zasubskrybować do odpytywania za pomocą funkcji ***client.subscribe()***.

Zainicjalizowanie urządzenia odbywa się za pośrednictwem funkcji ***setup()***. Jest ona odpowiedzialna za uruchomienie następujących elementów:

* **setup\_wifi()** (wcześniej opisana)
* Parametry potrzebne do połączenia z serwerem MQTT: adres serwera oraz port są podawane do funkcji ***client.setServer()*** i dzięki niej zestawiane jest połączenie.
* Funkcja ***client.setCallback()*** jest używana gdy nowa wiadomość dotarła do klienta, do niej jest podany wskaźnik  na funkcję callback.

Przesyłanie danych po magistrali I2C jest ustawiane przez funkcję ***Wire.begin()*** oraz ***dht.setup()***, której podany jest rodzaj czujnika i port. Dodatkowo, żeby prawidłowo działało I2C na ESP8266 funkcja ***ccs811.set\_i2cdelay()*** koryguje opóźnienia zegara.

Czujnik jest resetowany i identyfikowany:   
***bool ok= ccs811.begin();***

Poniższa funkcja uruchamia zbieranie pomiarów w odpowiednim trybie:  
***ok= ccs811.start(CCS811\_MODE\_1SEC);***

Pobieranie danych z czujnika CCS811 jest możliwe dzięki funkcji ***read()***, która odczytuje zbierane dane na temat zanieczyszczenia oraz liczby cząsteczek zawieszonych. Jeżeli chodzi o drugi czujnik - DHT22 umożliwione jest przez funkcje ***set\_envdata()***, która pobiera odczytaną temperaturę oraz wilgotność. Dodatkowo, otrzymywana wilgotność oraz temperatura były zabezpieczone przez funkcje ***assert()***, która zwraca błąd jeśli dane nie są zbierane. Kolejnym krokiem było zapisanie pobranych danych w formie łańcucha znaków, a następnie połączono temperaturę, wilgotność, stężenie cząsteczek oraz zanieczyszczenie w jeden ciąg znaków (***string***). Było to konieczne ze względu na serwer MQTT, który wymagał specyficznej formy dostarczanych danych.

W przypadku, kiedy moduł Wi-Fi nie został połączony z serwerem, wtedy zostaje wykonana funkcją **reconnect(),** która łączy się z  modułem po raz kolejny. Wysyłanie danych na serwer następuje jeżeli różnica pomiędzy czasem wysłania poprzedniej wiadomości, a czasem obecnym jest większa o 2 sekundy. Pobieranie czasu jest zapewnione przez funkcję ***millis()***. Następnie do zmiennej przypisywany jest czas wysłania poprzedniej wiadomości, a na ekran wypisywane są wartości odczytanych danych, wraz z komunikatem dla serwera MQTT.

**publish(0X, dane)**

Funkcją odpowiedzialną za przesłanie danych jest **publish()** z parametrami - “0X” oraz “dane” gdzie X jest numerem czujnika, a dane są pobranymi wartościami z czujników.

Pobieranie danych z czujnika oraz przesyłanie tych danych na server znajduję się w obrębie jednej funkcji loop, która wykonywana jest tak jak nazwa wskazuje - w pętli. Wykonywana ona jest co 2 sekundy, ale wartość ta może zostać zmieniona przez delay, na końcu funkcji.

# 4. Podsumowanie

Po poprawnym zaimplementowaniu i ustawieniu odpowiednich parametrów odpowiedzialnych za wysyłanie danych z czujników oraz poprawnego adresu IP wraz nazwą i hasłem dostępu system jest możliwe wysyłanie zebranych danych na serwer. Każdy z przygotowanych zestawów sensorów spełnia założone cele. Odbierane dane są poprawne oraz potwierdzone przez niezależny miernik temperatury, wilgotności oraz czujnik stężenia PM10.

# 5. Pomiary

Pomiary zostały przeprowadzone na dwóch czujnikach oddalonych od siebie około 50 cm. Przed rozpoczęciem testu sprawdzone zostały pomiary, które wskazywały na (względnie) taką samą temperaturę, wilgotność, a poziom zanieczyszczenia oraz stężenia PM10 oscylował na podobnym poziomie. Do sprawdzenia wpływu wilgotności na stężenie PM10 użyty został wysoce profesjonalny sprzęt generujący parę wodną. Pierwszy pomiar został rozpoczęty w momencie, kiedy urządzenie zaczęło generować parę, a czujnik odpowiedzialny za pomiar stężenia PM10 znalazł się tuż nad strumieniem pary. Wynikiem było około 10-krotnie wyższe stężenie cząsteczek PM10 niż na czujniku oddalonym 50 cm od strumienia pary wodnej, który wskazywał około 50  µg/m3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pomiar** | **Temperatura [°C]** | **Wilgotność [%]** | **Zanieczyszczenie** | **PM10 [ µg/m3]** |
| **0** | **23,2** | **33 %** | **???** | **50** |
| 1 | **23,4** | **42 %** | **???** | **562** |
|  |  |  |  |  |

MQTT Broker & Firebase Manager

# Streszczenie

Zrealizowany program pełni funkcję serwera i MQTT Brokera na platformie Raspberry PI3 z systemem operacyjnym Win 10 IoT. Serwer odbiera informacje z sensorów za pomocą protokołu MQTT i przesyła je do bazy danych czasu rzeczywistego dostarczonej przez Google, czyli Firebase.

# MQTT Broker

## Użyte biblioteki

MQTTnet – biblioteka .NET o wysokiej wydajności implementująca podstawową kominukację przez protokół MQTT. Zapewnia wsparcie zarówno dla klienta jak i serwera.

Główne cechy:

* Wsparcie operacji asynchronicznych
* Niskopoziomowa implementacja MQTT (Lightweight)
* Optymalizacja wydajności
* Wbudowane testy jednostokowe

Wspierane Frameworki:

* .NET Standard 1.3+
* .NET Core 1.1+
* .NET Core App 1.1+
* .NET Framework 4.5.2+ (x86, x64, AnyCPU)
* Mono 5.2+
* Universal Windows Platform (UWP) 10.0.10240+ (x86, x64, ARM, AnyCPU, Windows 10 IoT Core)
* Xamarin.Android 7.5+
* Xamarin.iOS 10.14+

## Kompilacja i działanie Brokera

MQTT Broker został zbudowany na platformę .NET Core i procesory ARM. Kod aplikacji jest dostępny na github`ie.

Krótki opis fukncji:

* StartBroker() – Uruchomienie brokera
* MqttServer\_ApplicationMessageRecevied() – event handler odpowiedzialny za odbiór wszystkich wiadomości wysłanych do servera MQTT
* MqttServer\_ClientConnected() – event handler odpowiedzialny za podłączenie do servera nowych klientów
* ConvertPayloadToDouble() – konwersja otrzymanego payloadu wiadomości z czujnika na tablice zmiennych typu double.

# Firebase Manager

DLL (Dynamic linked library), czyli biblioteka odpowiedzialna za przesyłanie danych z serwera do bazy danych w chmurze google`a czyli Firebase.

## Użyte biblioteki

Firesharp – Firebase REST API wrapper dla platform .NET i Xamarin.

Główne feature`y:

* SET
* PUSH
* GET
* Update
* Delete
* PushTaskAsync

Firesharp zapewnia pełną obsługę bazy danych z poziomu aplikacji opartej o .NET.

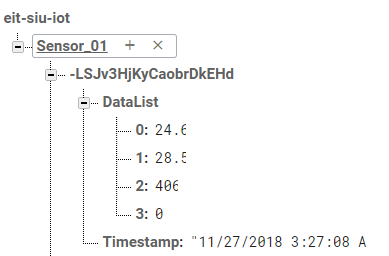
## Kompilacja i działanie Firebase Manager

Firebase Manager został zbudowany na platformę .NET Core i procesory ARM. Kod aplikacji jest dostępny na github`ie.

Krótki opis (ze względów bezpieczeństwa dokument nie będzie zawierał fragmentów kodu źródłowego):

Firebase Manager łączy się do Firebase przy u zyciu uniklanego klucza przypisanego bezpośrednio do aplikacji. Wszystkie wiadomości są wysyłane asynchronicznie dzięki czemu nie ma problemów z gubieniem wiadomości.

## Schemat



* Eit-siu-iot – nazwa bazy danych
* Sensor\_ID – sensor o konkretnym ID
* Uniklany klucz wpisu
* Lista danych i Timestamp

# Podsumowanie

Obie aplikacje są zainstalowane na Raspberry Pi 3, z systemem oprecyjnym Win 10 IoT. Serverem można zarządzać bezpośrednio z raspberry lub przez protokół SSH.

Aplikacje natywnie wyświetłają wszyskie wiadomości i zdarzenia, w celu wykrycia ewentualnych błędów.

Serwis internetowy

# Wizualizacja

## Użyte biblioteki

**Bootstrap** – framework zawierający zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS i może być stosowany m.in. do stylizacji takich elementów jak teksty, formularze, przyciski, wykresy, nawigacje i innych komponentów wyświetlanych na stronie. Framework korzysta także z języka JavaScript. Zapewnia wsparcie stron responsywnych oraz optymalizację wydajności.

**Google Charts** – API firmy Google pozwalające dynamicznie tworzyć wykresy na stronach WWW z dowolnych danych. Licencja pozwala na wykorzystanie narzędzia do dowolnych celów (również komercyjnych), bez ograniczonego transferu.

**Firebase API** – interfejs platformy Firebase, w tym przypadku do języka JavaScript. Pozwala łączyć się i pobierać dane z rzeczywistej bazy danych.

Testowane przeglądarki:

* Google Chrome
* Microsoft Edge
* Mozilla Firefox

## Parsowanie i działanie

Serwis został napisany głównie w JavaScript, ale również w PHP, HTML i CSS.

Krótki opis funkcjonalności:

* W pliku index.php tworzony jest ogólny zarys strony oraz dodawane są niezbędne odnośniki do bibliotek, w pliku style.css definiowane są style, następnie w pliku charts.php:
* Za pomocą metody $\_GET pobierany jest parametr globalny $dwmyChoose, determinujący okres widoczności danych
* Konfigurowane, a następnie nawiązywane zostaje połączenie z serwerem Firebase
* Pobierane są wstępne adresy danych (numer sensora i odpowiadające mu odczyty) i układane w tablicę adresową
* Dane są odczytywane i układane na podstawie podziału czasowego, podziału na czujniki i poszczególne wartości
* Pobierana są dane o ostatnich odczytach i porównywane z normami zanieczyszczenia PM10
* Ładowane i wypełniane danymi są moduły wykresów Google Charts

Przykładowe funkcje:

* firebase.database().ref()
* getTime(), getDate()
* getElementById()
* setAttribute()

# Podsumowanie

Serwisem można zarządzać z poziomu przeglądarki internetowej lub dowolnego klienta FTP. Aplikacja wyświetla rzeczywiste dane zgromadzone na bazie Firebase od dnia obecnego cofając się w tył o wybrany okres, np. dla prezentacji tygodniowej przedstawione są dane z poprzednich siedmiu dni. Serwis jest dostępny pod tymczasowym adresem: <http://niezapominajka.info/PoCoZakladacNowyHostingSkoroMoznaSkorzystacZIstniejacegoBezReklamTylkoWsadzicBardzoDlugiLinkNaKtoryNiktNigdyNieWpadnie/>