# WOJSKOWA AKADEMIA TECHNICZNA

**im. Jarosława Dąbrowskiego**

## WYDZIAŁ CYBERNETYKI



# PRACA DYPLOMOWA

## STACJONARNE STUDIA I°

Temat pracy: **SYSTEM GROMADZENIA I ANALIZY DANYCH SENSORYCZNYCH Z URZĄDZEŃ IOT Z WYKORZYSTANIEM PRZETWARZANIA W CHMURZE**

**INFORMATYKA**

(kierunek studiów)

**SYSTEMY INFORMATYCZNE**

(specjalność)


Dyplomant: **Dominik DAWIDZIAK**

Promotor pracy: **dr inż. Michał DYK**



**Warszawa 2021**

**OŚWIADCZENIE**

*„Wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum WAT”.*

Dnia ..... .....
(podpis)

*Pracę przyjąłem*

*promotor pracy*
*dr inż. Michał Dyk*

*Zadanie do pracy dyplomowej – str. 1*

***Zadanie do pracy dyplomowej – str. 2***

# SPIS TREŚCI

WSTĘP ..... 8
ROZDZIAŁ I. ..... 10
CHARAKTERYSTYKA DZIEDZINY ..... 10
1.1. INTERNET RZECZY ..... 10
1.1.1. Technologie ..... 10
1.1.2. Zastosowanie ..... 11
1.1.3. Standardy ..... 13
1.1.4. Architektura rozwiązań IoT ..... 14
1.1.5. Protokoły komunikacyjne ..... 17
1.1.6. Platformy sprzętowe ..... 21
1.2. PRZETWARZANIE W CHMURZE ..... 24
1.2.1. Oferowane usługi ..... 25
1.2.2. IoT w chmurze ..... 25
1.2.3. Komunikacja z chmurą ..... 27
1.2.4. Stos technologiczny platformy IoT ..... 27
1.2.5. Przegląd dostępnych platform IoT ..... 28
ROZDZIAŁ II. ..... 32
PROJEKT SYSTEMU GROMADZENIA I ANALIZY DANYCH SENSORYCZNYCH Z URZĄDZEŃ IOT ..... 32
2.1. Opis systemu ..... 32
2.2. Wymagania ..... 32
2.2.1. Funkcjonalne ..... 32
2.2.2. Pozafunkcjonalne ..... 44
2.3. Architektura rozwiązania ..... 44
2.3.1. Architektura wysokopoziomowa ..... 44
2.3.2. Architektura szczegółowa ..... 46
2.4. Projekt platformy ..... 48
2.4.1. Komponenty systemu ..... 48
2.4.2. Aplikacja internetowa ..... 53
2.4.3. Specyfikacja REST API ..... 57

ROZDZIAŁ III. ..... 59
IMPLEMENTACJA SYSTEMU ..... 59
3.1. Narzędzia ..... 59
3.2. Implementacja wybranych komponentów systemu ..... 60
3.2.1. Przygotowanie przestrzeni chmurowej ..... 60
3.2.2. Utworzenie serwisu IoT Hub ..... 60
3.2.3. Stream Analytics oraz Table Storage ..... 62
3.2.4. Program uruchamiany przez urządzenie IoT ..... 63
3.2.5. Web API ..... 67
3.2.6. Aplikacja kliencka (React Single Page Application) ..... 71
3.2.7. Azure Active Directory ..... 71
3.2.8. Azure App Services i publikacja aplikacji ..... 74
ROZDZIAŁ IV. ..... 75
TESTY SYSTEMU W RAMACH WYBANEGO CASE STUDY ..... 75
4.1. Przypadek testowy ..... 75
4.2. Scenariusz przeprowadzenia testu ..... 75
4.3. Testy systemu ..... 76
4.3.1. Zakładanie nowego konta ..... 76
4.3.2. Dodanie nowych urządzeń ..... 78
4.3.3. Pobranie plików konfiguracyjnych dla nowych urządzeń ..... 81
4.3.4. Wgranie plików konfiguracyjnych do pamięci urządzeń ..... 82
4.3.5. Rozpoczęcie gromadzenia danych ..... 82
4.3.6. Wylogowanie z systemu ..... 85
4.3.7. Logowanie do systemu ..... 85
4.3.8. Modyfikacja konfiguracji jednego z urządzeń ..... 85
4.3.9. Podgląd zgromadzonych danych ..... 86
4.3.10. Podgląd analizy danych ..... 87
4.3.11. Usunięcie urządzeń ..... 90
4.4. Wnioski ..... 90
ZAKOŃCZENIE ..... 91
BIBLIOGRAFIA ..... 92

Spis rysunków ..... 95
Spis tabel ..... 97
Spis listingów ..... 98

# WSTĘP

W dobie nieprzerwanego dostępu do informacji, gdy niemal 3 miliardy osób na Ziemi posiada smartfon lub smartwatch a prawie 4 miliardy osób korzysta z Internetu, ilości danych jakie są generowane, przetwarzane i analizowane w ciągu jednego dnia szacowane są na 2,5 tryliona bajtów. Za wygenerowanie 90% tych danych odpowiadają jednak urządzenia Internetu Rzeczy (IoT – ang. Internet of Things)[1].

Od roku 2006 kiedy to liczba rzeczy typu *smart* działających w sieci była szacowana na 2 miliardy, na przestrzeni 14 lat, nastąpił co najmniej stukrotny wzrost liczby urządzeń IoT podłączonych do Internetu, Szacuje się że około 40% z 200 miliardów urządzeń IoT wspiera już sektor przemysłowy (procesy technologiczne, łańcuchy dostaw), 30% sektor medyczny a 8% handlowy[1].

Tak szybki rozwój w dziedzinie Internetu Rzeczy wymusza potrzebę szybkiego odpowiadania na wiele, coraz to większych potrzeb, wynikających z cięgle narastających ilości danych. Potrzebna jest ogromna przestrzeń dyskowa pozwalająca na magazynowanie wielkich zbiorów danych, wielka moc obliczeniowa dająca możliwość analizy, a także infrastruktura pozwalająca na zarządzanie setkami a nawet tysiącami urządzeń działającymi w ramach określonego systemu.

Technologią sprawnie odpowiadająca na potrzeby Internetu Rzeczy jest technologia chmurowa, która z dnia na dzień coraz bardziej wypiera tzw. rozwiązania on-premises[2].

W niniejszej pracy został przedstawiony cały proces projektowania, implementacji i testowania systemu opartego o technologie chmurowe, pozwalającego na gromadzenie i analizę danych generowanych przez urządzenia IoT.

Zadania do pracy zostały zrealizowane w rozdziałach:

- Charakterystyka dziedziny

Wprowadzającym w technologię Internetu Rzeczy oraz przetwarzania chmurowego.

- Projekt Systemu Gromadzenia Danych Sensorycznych z urządzeń IoT

Opisującym budowę systemu od strony technicznej.

- Implementacja systemu

Pokazującym jak przebiegała i jak została przeprowadzona implementacja zaprojektowanego systemu.

- Testy systemu w ramach wybranego CASE STUDY

Pokazującym funkcjonalności systemu na przykładzie przeprowadzania przykładowego scenariusza przypadku testowego.

Głównym założeniem pracy jest realizacja budowy systemu działającego w oparciu o technologie chmurowe. System ten, ma odpowiedzieć na potrzeby wynikające z gromadzenia danych generowanych przez urządzenia IoT. Utworzone na przestrzeni pisania tej pracy rozwiązanie, ma być gotową do wykorzystania przez wielu potencjalnych użytkowników, platformą udostepniająca możliwości zarządzania wieloma urządzeniami IoT oraz analizy zgromadzonych danych.

# ROZDZIAŁ I.

# CHARAKTERYSTYKA DZIEDZINY

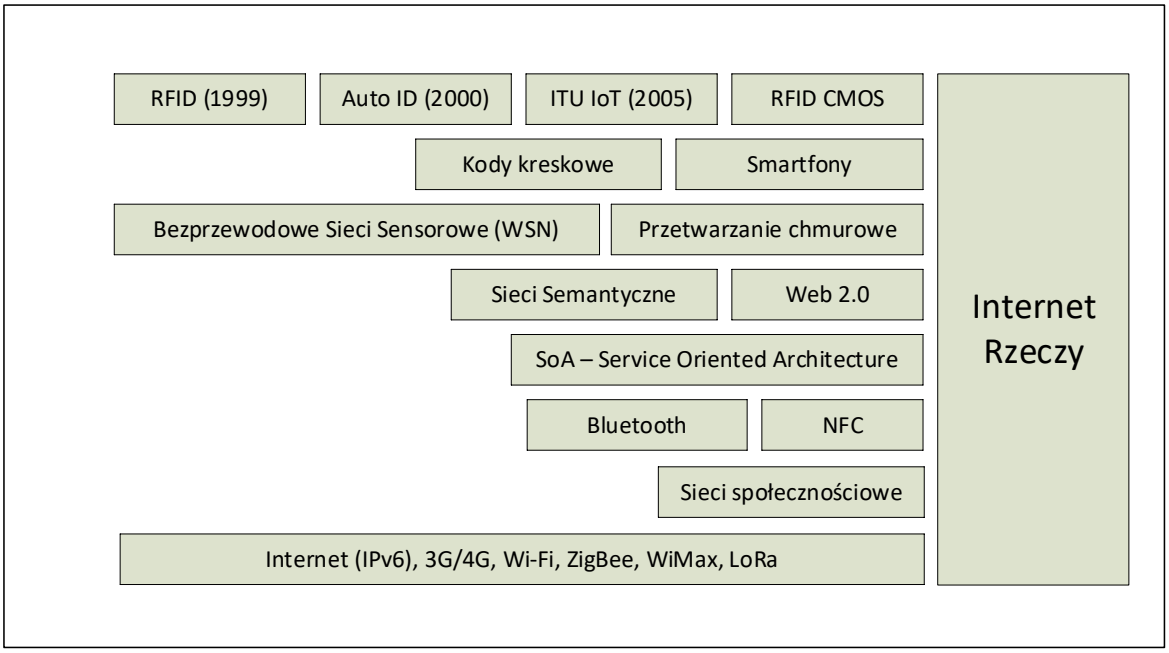
## 1.1. INTERNET RZECZY

Pojęcie Internetu Rzeczy (ang. *Internet of Things* – w skrócie *IoT)* używane jest w odniesieniu do urządzeń wyposażonych w różnego rodzaju sensory, podłączonych do sieci, komunikujących się i przetwarzających informacje[3]. Urządzeniem Internetu Rzeczy może być nazwane niemal każde urządzenie z jakim mamy do czynienia na co dzień, począwszy od używanych przez nas na co dzień smartfonów i smartwatchów, po urządzenia AGD które dzięki wbudowanym czujnikom i połączeniu z siecią mogą dostarczać nam danych o badanej wielkości fizycznej np. temperaturze.

W znacznym uproszczeniu można powiedzieć, że każde urządzenie, które generuje dane, łączy się z Internetem i można je włączyć i wyłączyć, może zostać uznane za urządzenie IoT[4].

### 1.1.1. Technologie

Rozwój komunikacji bezprzewodowej znacznie rozszerzył możliwości Internetu Rzeczy. Internet Rzeczy gromadzi wokół siebie obszerny stos technologiczny, Bezprzewodowe Sieci Sensorowe (WSN), Radiowa Identyfikacja Obiektów (RFID) – z której to tak naprawdę Internet Rzeczy się wywodzi, systemy komunikacji bezprzewodowej takie jak Bluetooth czy Wi-Fi, a także przetwarzanie chmurowe które w ostatnich latach zyskało ogromną popularność.




Źródło: Opracowanie własne w Microsoft Visio

**Rys. 1. Technologie powiązane z IoT**[3]

### 1.1.2. Zastosowanie

Urządzenia IoT znajdują zastosowanie w kluczowych procesach i rozwiązaniach w wielu dziedzinach życia. Wyróżnić można następujące obszary, wraz z przykładami zastosowań Internetu Rzeczy jak:

- Budynki

Budynki przemysłowe, hale produkcyjne, supermarkety czy lotniska. W budynkach typu *Smart* urządzenia IoT sosowane są w celu zapewnienia lub zwiększenia bezpieczeństwa (np. systemy przeciwpożarowe), sterowania oświetleniem czy kontroli dostępu[5].

- Energetyka

Przemysł gazowy/naftowy, alternatywne źródła energii, dostarczanie i zapewnienie stałego dopływu energii oraz praca elektrowni konwencjonalnych. W tych gałęziach przemysłu energetycznego urządzenia IoT odpowiadają za np. nadzorowanie pracy turbin wiatrowych, awaryjnych systemów zasilania czy procesów wydobycia węgla[5].

- Gospodarstwa domowe/sektor konsumencki

Konsumenci korzystają z urządzeń IoT do sterowania automatyką domową, monitorowania temperatury w pomieszczeniach, prowadzenia kontroli rodzicielskiej czy też do rozrywki[5].

- Opieka zdrowotna

Stosowanie rozwiązań IoT w przemyśle medycznym w dużym stopniu usprawnia prace personelu medycznego.

Dzięki Internetowi Rzeczy powstały takie rozwiązania jak: wszczepiane rozruszniki serca, mogące na odległość monitorować stan pacjenta, różnego rodzaju sprzęt medyczny – inkubatory, pompy przeciwbólowe czy urządzenia laboratoryjne, które mogą pracować autonomicznie, bez potrzeby nieprzerwanego nadzoru człowieka[5].

- Przemysł

Monitorowanie łańcuchów dostaw, stanów magazynowych, nadzorowanie pracy urządzeń przemysłowych czy przebiegu procesów produkcyjnych – to zaledwie kilka z szeregu przykładów zastosowań urządzeń IoT w szeroko pojętej branży przemysłowej[5].

- Transport

Coraz częściej pojazdy a także inne środki transportu (samoloty, pociągi itp.) zaopatrywane są w inteligentne systemy diagnostyczne, systemy nawigacji i inne rozwiązania, mające zwiększyć bezpieczeństwo oraz komfort ich użytkowania[5].

- Handel

W sektorze handlu urządzenia IoT znajdują zastosowania przy monitorowaniu przesyłek, gromadzeniu informacji o konsumentach oraz produktach, kontrolowaniu stanów magazynowych jak i również w automatach z żywnością i napojami czy automatach do gier[5].

- Bezpieczeństwo

Wspomaganie służb ratunkowych, policji, straży pożarnej czy wojska. Monitorowanie środowiska (jakość powietrza, wielkość opadów), nadzór (systemy monitoringu, fotoradary)[5].

- Informatyka i telekomunikacja

Sieci korporacyjne, centra danych, sieci prywatne, zdalne monitorowanie infrastruktury sieciowej czy też zarządzanie systemami zasilania czy klimatyzacji pomieszczeń serwerowych[5].

### 1.1.3. Standardy

Wprowadzenie i przestrzeganie określonych standardów znacznie ułatwia proces projektowania i budowania rozwiązań. W ostatnich latach wiele światowych organizacji (takich jak np. IEEE - ang. *Institute of Electrical and Electronics Engineers* - Instytut Inżynierów Elektryków i Elektroników) rozwinęło wiele standardów, które mają istotne znaczenie dla Internetu Rzeczy[3].

**Tab. 1. Standardy komunikacyjne i sensoryczne znajdujące zastosowanie dziedzinie Internetu Rzeczy.**

| Technologia | Standardy |
|---|---|
| Komunikacja | IEEE 802.15.4 (ZigBee) |
| | IEEE 802.11 (WLAN) |
| | IEEE 802.15.1 (Bluetooth) |
| | IEEE 802.15.6 (WBAN) |
| | IEEE 1888 |
| | IPv6 |
| | 3G/4G |
| | UWB |
| Sensory | ISO/IEC JTC1 SC31 i ISO/IEC |
| | JTC1 WG7 |
| | IEEE 1452.x, IEC S.C. 17B, EPC global, ISO TC 211, ISO TC 205 |

Źródło: [3]

### 1.1.4. Architektura rozwiązań IoT

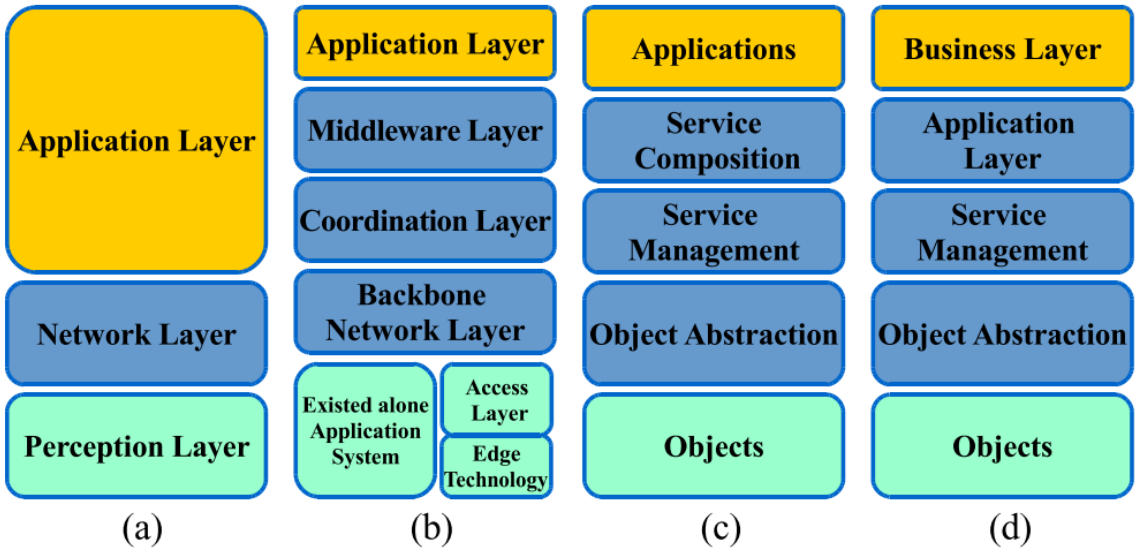
Żaden z obecnie istniejących standardów nie jest dominujący w kwestii architektury rozwiązań IoT, warto jednak zaznaczyć, że preferowanym podejściem jest architektura warstwowa[6].

Trwają prace nad projektami (np. IoT-A – Internet of Things Architecture), które miałby zunifikować architekturę tworzonych rozwiązań, poprzez zdefiniowanie powszechnego modelu odniesienia będącego w stanie odpowiedzieć na potrzeby napływające od naukowców czy przemysłu[7].

Istnieją swego rodzaju wzorce, które na dzień dzisiejszy, są szeroko wykorzystywane przy tworzeniu rozwiązań IoT, podstawowym z nich jest model Trójwarstwowy, oparty o następujące składowe:

- Warstwa Aplikacji (ang. *Application Layer)* – odpowiadająca za analizę i wizualizację zebranych danych,
- Warstwa Sieci (ang. *Network Layer)* – umożliwiająca przepływ danych,
- Warstwa Percepcji (ang. *Perception Layer)* – skupiająca w sobie brzegowe urządzenia IoT odpowiedzialne za generowanie danych na podstawie przeprowadzanych pomiarów.

Na przestrzeni czasu pojawiły się kolejne propozycje wzorców architektonicznych, często opatrzonych większą liczbą abstrakcji[6]. Rysunek 2. przedstawia kilka z nich. W celu omówienia idei architektury warstwowej w rozwiązaniach IoT posłużę się zaproponowaną architekturą Pięciowarstwową.




Źródło: [6]

**Rys. 2. Architektury rozwiązań IoT. (a) Trójwarstwowa (ang. *Three-layer*). (b) Bazująca na Warstwie Pośredniej (ang. *Middleware based*). (c) Zorientowana na usługi (ang. *SOA based*). (d) Pięciowarstwowa (ang. *Five-layer*)**[6]**.**

- Warstwa Obiektów

Pierwsza warstwa, może być też nazywana Warstwą Percepcji. Są to fizyczne urządzenia wyposażone w sensory, których zadaniem jest zbieranie i przetwarzanie informacji. To właśnie ta warstwa zapewnia takie funkcjonalności rozwiązań IoT jak udostępnianie danych o lokalizacji, temperaturze, ruchu, przyspieszeniu czy wilgotności powietrza. W niej wielkość fizyczna zamieniana jest na sygnał cyfrowy.

W Warstwie Percepcji powinien być obecny ustandaryzowany mechanizm *Plug-and-Play*[6], który pozwoliłby na łatwe i szybkie podłączanie nowych urządzeń końcowych IoT (często różnych i badających różne wielkości fizyczne) do działającego systemu.

- Warstwa Abstrakcji

Abstrakcja ukrywa operacje które są wykonywane podczas przesyłania danych wytworzonych przez urządzenia końcowe do Warstwy Zarządzania Usługami. Dane mogą być przesyłane na pomocą takich technologii jak RFID, 3G, GSM, UMTS, Wi-Fi itp.[6]. Procesy zarządzania danymi i przetwarzania chmurowego również odbywają się w tej warstwie.

- Warstwa Zarządzania Usługami

Może być też nazywana Warstwą Pośrednią, odpowiada za realizacje otrzymywanych zapytań, pozwala programistą na prace z heterogeniczną platformą bez konieczności skupiania się na konkretnym sprzęcie (urządzeniu) wykorzystanym w rozwiązaniu[8]. Ta warstwa dostarcza wyspecjalizowane serwisy charakterystyczne dla architektury typu SOA.

- Warstwa Aplikacji

Warstwa zapewniająca usługi dla końcowego użytkownika systemu np. odczyty temperatury lub wilgotności powietrza. Istotność tej warstwy wynika z faktu, że to ona zapewnia wysokiej jakości informację o badanych przez urządzenia IoT wielkościach[6].

- Warstwa Biznesowa

Warstwa odpowiedzialna za wizualizacje i raportowanie zebranych i przetworzonych danych w postaci przyjaznej użytkownikowi (wykresy, grafy, diagramy). Ta warstwa bezpośrednio wspiera procesy decyzyjne użytkownika końcowego, opierane na analizie zebranych danych. Poza opisanymi funkcjonalnościami także monitorowanie i zarządzanie warstwami niższymi odbywa się również na tym poziomie[6].

Większość, popularnych obecnie, rozwiązań chmurowych pokrywa się z tą propozycją architektury. Wydaje się ona być również bardzo korzystna z punktu widzenia ograniczeń sprzętowych mogących występować po stronie urządzeń brzegowych. Obciążenia które mógłby pojawić się w czasie przetwarzania i magazynowania danych w Warstwie Percepcji są przenoszone do warstw wyższych, często wieloprocesorowych serwerów lub centrów danych udostępniających usługi chmurowe.

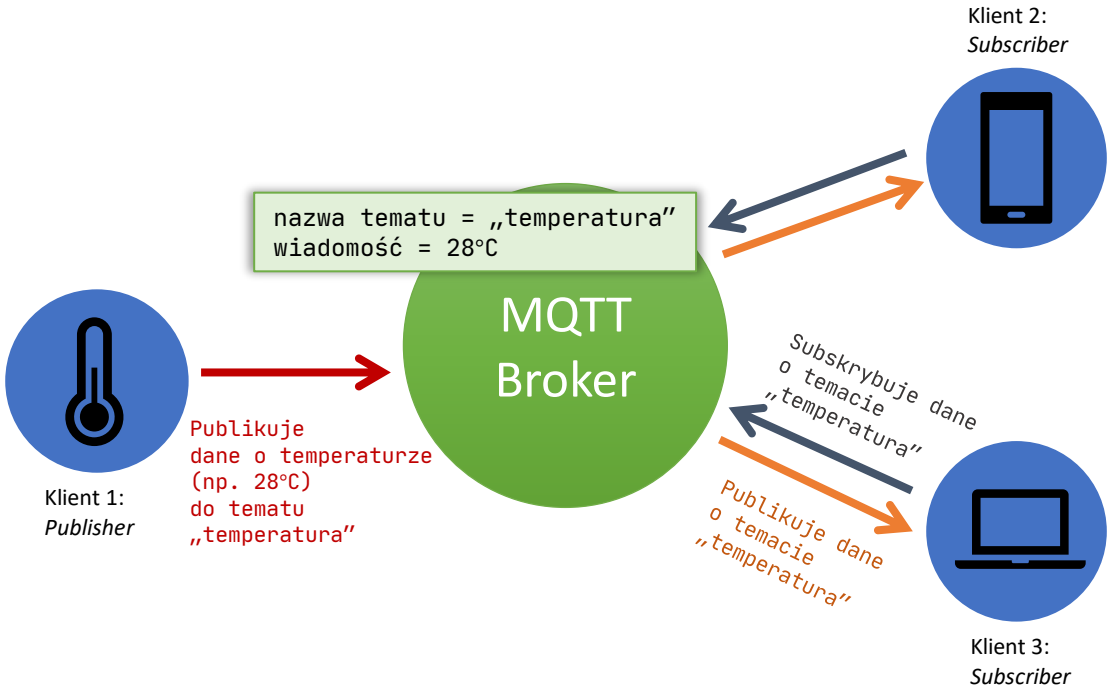
### 1.1.5. Protokoły komunikacyjne

Protokoły komunikacyjne umożliwiają wymianę informacji pomiędzy dwoma punktami, najczęściej Klient – Serwer lub Klient – Klient.

- MQTT

MQTT (Message Queue Telemetry Transport) – utworzony przez IBM protokół transmisji danych, oparty na mechanizmie publikacja-subskrypcja. Zapewnia niezawodną transmisję danych przy małym zużyciu zasobów sprzętowych urządzeń[9].

Wykorzystywanie takiego protokołu komunikacji jest zasadne i trafne w rozwiązaniach IoT, zwłaszcza tam gdzie pojawia się niestabilne połączenie.




Źródło: Opracowanie własne

**Rys. 3. Model przesyłania wiadomości za pomocą protokołu MQTT - przykład dla danych o temperaturze**

Architektura protokołu MQTT opiera się na trzech zasadniczych komponentach:

- Publisher
- Broker

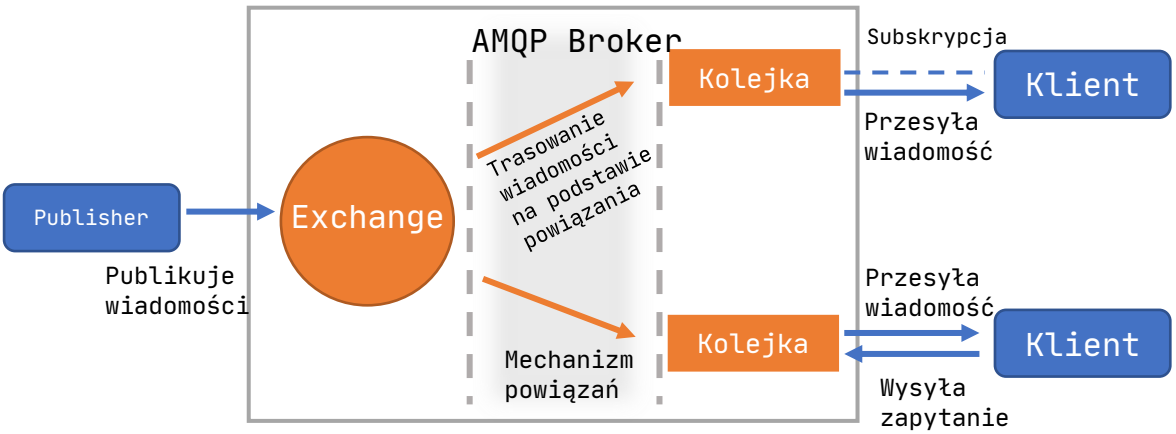
- Subscriber

Informacje generowane przez urządzenia IoT są przesyłane do Brokera (Pośrednika) który udostępnia informacje do subskrypcji urządzeniom klienckim. Klient ma możliwość subskrybowania informacji dotyczących interesującego go tematu.

Bezpieczeństwo tego protokołu zapewnione jest poprzez weryfikowanie urządzeń publikujących dane oraz subskrybentów w warstwie pośredniczącej[10]

- AMQP

AMQP (Advanced Message Queuing Protocol) – protokół komunikacyjny zorientowany na komunikaty, będący otwartym standardem. Jego głównymi wyróżnikami są: kolejkowanie wiadomości, routing, bezpieczeństwo i niezawodność.




Źródło: Opracowanie własne

**Rys. 4. Model przesyłania wiadomości za pomocą protokołu AMQP**

Broker w protokole AMQP składa się z trzech typów mechanizmów:

- Mechanizmu dzielenia (Exchange),
- Mechanizmu powiązań (Binding),
- Mechanizmu kolejkowania (Queues).

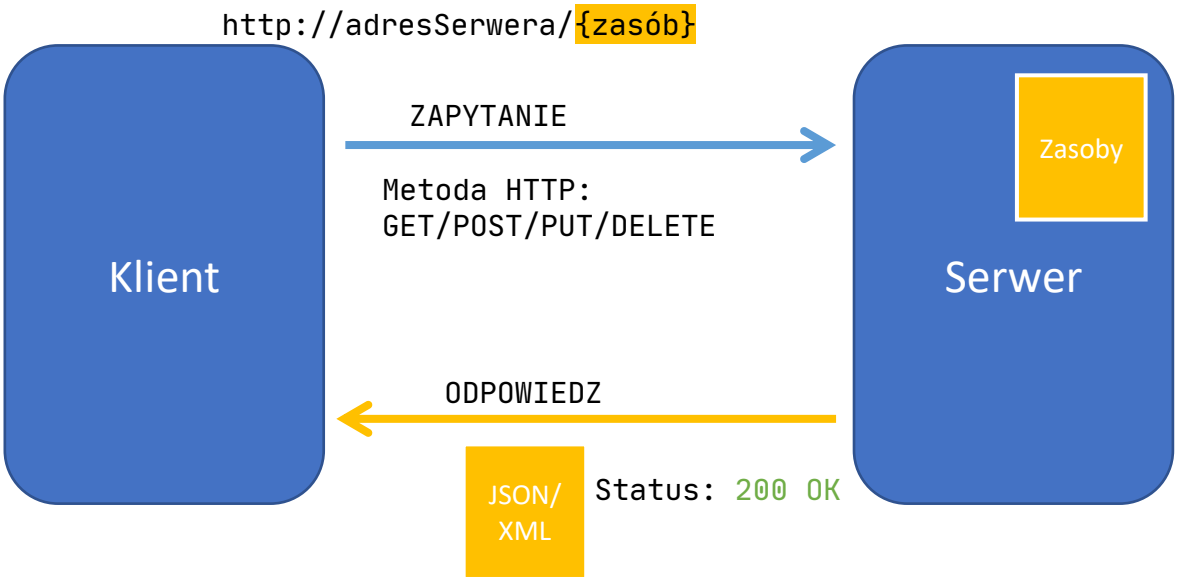
Wiadomości publikowane przez urządzenia końcowe przesyłane są do Brokera. Mechanizm Exchange rozdziela otrzymywane wiadomości do odpowiednich kolejek w zależności od zdefiniowanych powiązań.

Komunikacja między urządzeniem klienckim a Brokerem może być oparta o mechanizm powiadomień, subskrypcji jak i zapytań[9].

- HTTP

Sam protokół HTTP określa format komunikacji typy żądanie-odpowiedz. Aktualnie protokół ten jest bardzo mocno wykorzystywany przez przeglądarki internetowe, lecz z sukcesem odnajduje także swoje zastosowanie w dziedzinie Internetu Rzeczy.

Warto w tym momencie wspomnieć o REST (Representational State Transfer), który nie jest stricte protokołem, a jedynie zbiorem zasad (dobrych praktyk) składających się na wzorzec tworzenia interfejsów API, najczęściej na bazie protokołu HTTP (*Hypertext Transfer Protocol*). Stosowanie zasad REST zapewnia łatwe operowanie na zasobach, różne rodzaje operacji oraz ich bezstanowość, a także buforowanie danych[9][11].




Źródło: Opracowanie własne

**Rys. 5. Model przesyłania wiadomości za pomocą protokołu HTTP**

Klient korzystając z jednej z dostępnych metod HTTP wysyła żądanie do serwera. Żądanie jest powiązane z zasobem z pomocą identyfikatora URI (*Uniform Resource Identifier*), które klient umieszcza w adresie URL, pod który kierowane jest zapytanie. Forma przesyłanej wiadomości jest jasno określona:

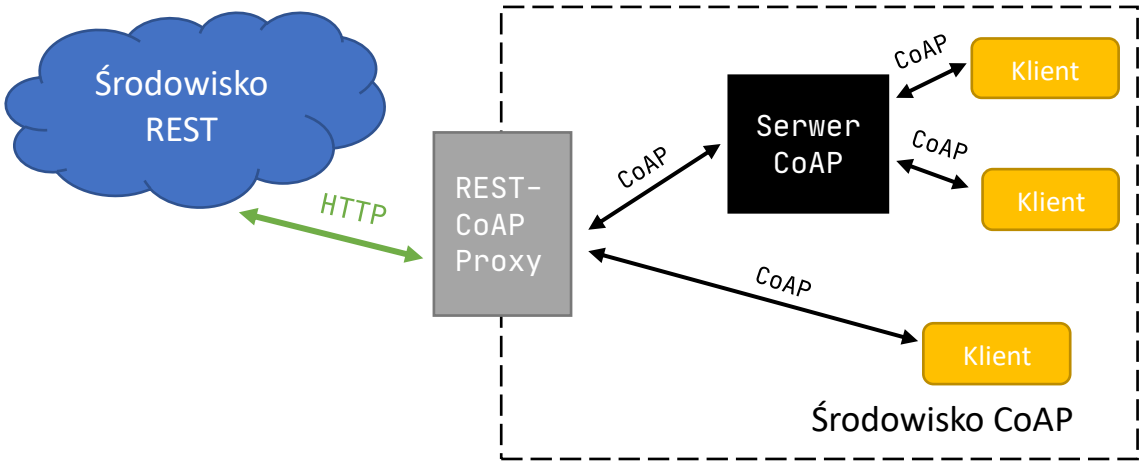
- Pierwsza linia wiadomości określa metodę HTTP, zasób i wersję wykorzystywanego protokołu,
- Kolejne linie zawierają nagłówki w formacie `nazwa nagłówka: wartość nagłówka`, które zawierają informacje o metadanych wiadomości,
- Pusta linia, mówiąca o końcu nagłówków,
- Ciało wiadomości tzw. *Body* jeżeli takowe istnieje.

Odpowiedz przesyłana przez serwer jest bardzo podobna, jedynie w pierwszej linijce nie ma metody HTTP ani nazwy zasobu, a pojawia się status odpowiedzi[11].

- CoAP

CoAP (Constrained Application Protocol) – protokół komunikacyjny zdefiniowany w oparciu o REST i funkcjonalności HTTP. W przeciwieństwie do HTTP transfer danych odbywa się w oparciu o protokół UDP (nie TCP) co czyni go bardziej odpowiednim dla urządzeń IoT – ze względu na ograniczone zasoby sprzętowe wykorzystywanych urządzeń[6].

Ograniczenia wynikające z rezygnacji ze śledzenia sesji, kontroli przepływu czy retransmisji danych, pozwalają na wykorzystanie tego protokołu w urządzeniach typu *low-power* a także na ograniczenie nadmiernej konsumpcji pamięci obliczeniowej.




Źródło: Opracowanie własne

**Rys. 6. Diagram współpracy środowiska CoAP z REST**

CoAP charakteryzuje się interoperacyjnością z rozwiązaniami opartymi o REST. Współpraca środowisk REST i CoAP najczęściej odbywa się na pomocą serwera pośredniczącego Proxy.

Wszystkie zaprezentowane protokoły, poza CoAP, oparte są o protokół TCP/IP. Należy również zaznaczyć, że opisane protokoły znajdują się w warstwie aplikacyjnej modelu odniesienia ISO/OSI.

W literaturze i źródłach internetowych można znaleźć, dużo więcej protokołów komunikacyjnych wykorzystywanych w rozwiązaniach Internetu Rzeczy. Nie mniej, na potrzeby charakterystyki dziedziny tej pracy skupiłem się wyłączne na czterech najbardziej popularnych, i najszerzej wykorzystywanych do współpracy z rozwiązaniami chmurowymi.

### 1.1.6. Platformy sprzętowe

Obecnie na rynku dostępny jest szeroki wachlarz urządzeń Internetu Rzeczy, począwszy od urządzeń wymagających jedynie podłączenia do istniejącego systemu bądź nawet działających autonomicznie, po zaawansowane otwarte platformy sprzętowe przeznaczone do tworzenia prototypów a następnie gotowych komercyjnych rozwiązań. Szczególną uwagę należy zwrócić na urządzenia typu mini-komputery czy płytki deweloperskie, które dzięki posiadanym zasobom pamięciowym i obliczeniowym nie tylko pozwalają deweloperowi zbudować urządzenie które sprosta stawianym wymaganiom ale także udostępniają praktycznie nieograniczone możliwości rozwoju, skalowalności czy modyfikowania projektu.

Na potrzeby niniejszej pracy opiszę dwie platformy sprzętowe charakteryzujące się wysoką dostępnością i popularnością wśród deweloperów IoT.

Pierwszą z nich jest Raspberry Pi, warto ją szerzej opisać, ponieważ posiada ona wsparcie wielu systemów operacyjnych, a co za tym idzie wiele języków programowania (takich jak Python, Java, Ruby, C, C++ czy C#) może zostać wykorzystane do tworzenia projektów w oparciu o tą platformę. Warto też wspomnieć o społeczności skupionej wokół Raspberry Pi, która tworzy i udostępnia ogromne ilości darmowych bibliotek a także zapewnia szeroko pojęte wsparcie dla deweloperów korzystającym z tej platformy[12].

Drugą platformą godną uwagi jest moduł ESP8266 w oparciu o który powstało wiele płytek deweloperskich charakteryzujących się bardzo niską ceną rzędu 15-25zł (Ceny na popularnym polskim serwisie aukcyjnym). Płytki te oferują znacznie mniejsze zasoby w porównaniu do mikrokomputerów Raspberry Pi natomiast tak samo skupiają wokół siebie ogromną społeczność udostępniającą duże zasoby otwartego oprogramowania[13].

- Raspberry Pi

Małe i bardzo popularne urządzenie  które często nazywane jest kieszonkowym komputerem. Po podłączeniu do niego klawiatury, myszy, monitora i kabla sieciowego, uruchomianiu na nim systemu operacyjnego, otrzymujemy niemal pełnoprawną jednostkę pozwalającą użytkownikowi na uruchamianie aplikacji czy przeglądanie Internetu. Mimo tych podstawowych funkcji, głównym przeznaczeniem Raspberry jest projektowanie i budowa urządzeń oraz systemów w oparciu o ten moduł[4].

Źródło: https://www.pi-shop.ch/raspberry-pi-3

**Rys.  7. Przykładowa płytka - Raspberry Pi 3 Model B**

Na rynku dostępnych jest wiele wersji minikomputera Raspberry Pi, każda z nich wyróżnia się nieco inną specyfikacją.

Przykładowa specyfikacja (model: Raspberry Pi Zero W)

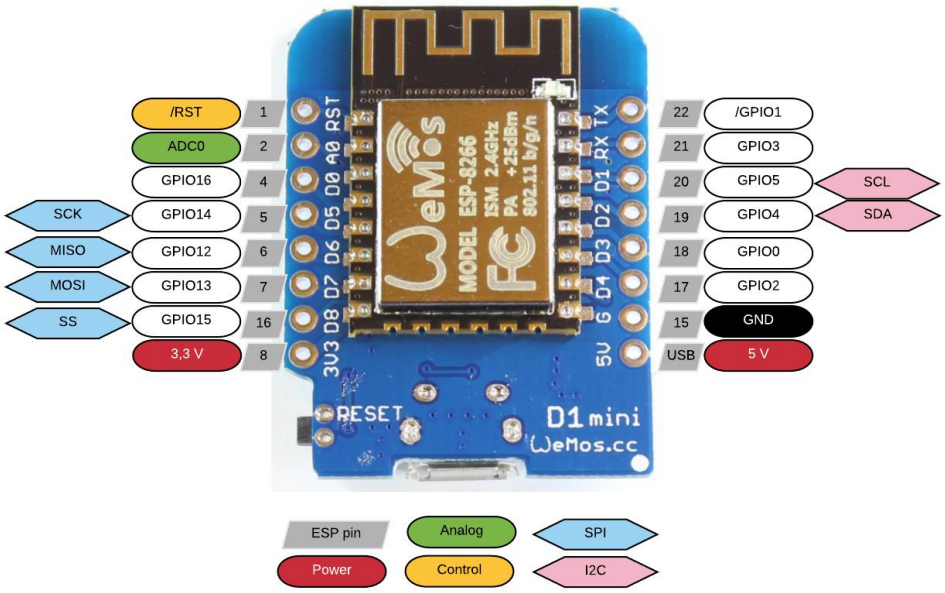
- Procesor: Broadcom BCM2835 ARM11 1GHz (1 rdzeń)
- Pamięć RAM: 512 MB

- Bluetooth Low Energy BLE 4.1
- Porty: miniHDMI, USB
- 40 x GPIO (general-purpose input/output)
- Interfejs WiFi: 802.11 b/g/n 150 Mbps
- Komunikacja: UART, SPI, I2C

Platforma najczęściej działa pod kontrolą systemu operacyjnego Linux – Raspberry Pi OS lub Windows 10 IoT[14].

- ESP8266

ESP8266 to bardzo popularny układ w oparciu o który można zbudować, urządzenie komunikując się za pomocą WiFi. Charakteryzuje się niewielkim rozmiarem, wysoką dostępnością i bardzo niską ceną. Układ integruje w sobie elementy wymagane do komunikacji WiFi w standardzie 802.11 b/g/n. Na rynku wiele urządzeń działa w oparciu właśnie o ten układ a także istnieje szeroki wybór gotowych modułów czy płytek rozwojowych z ESP8266 np. ESP-01, ESP-12E lub WeMos D1 przeznaczonych do tworzenia własnych rozwiązań[13].



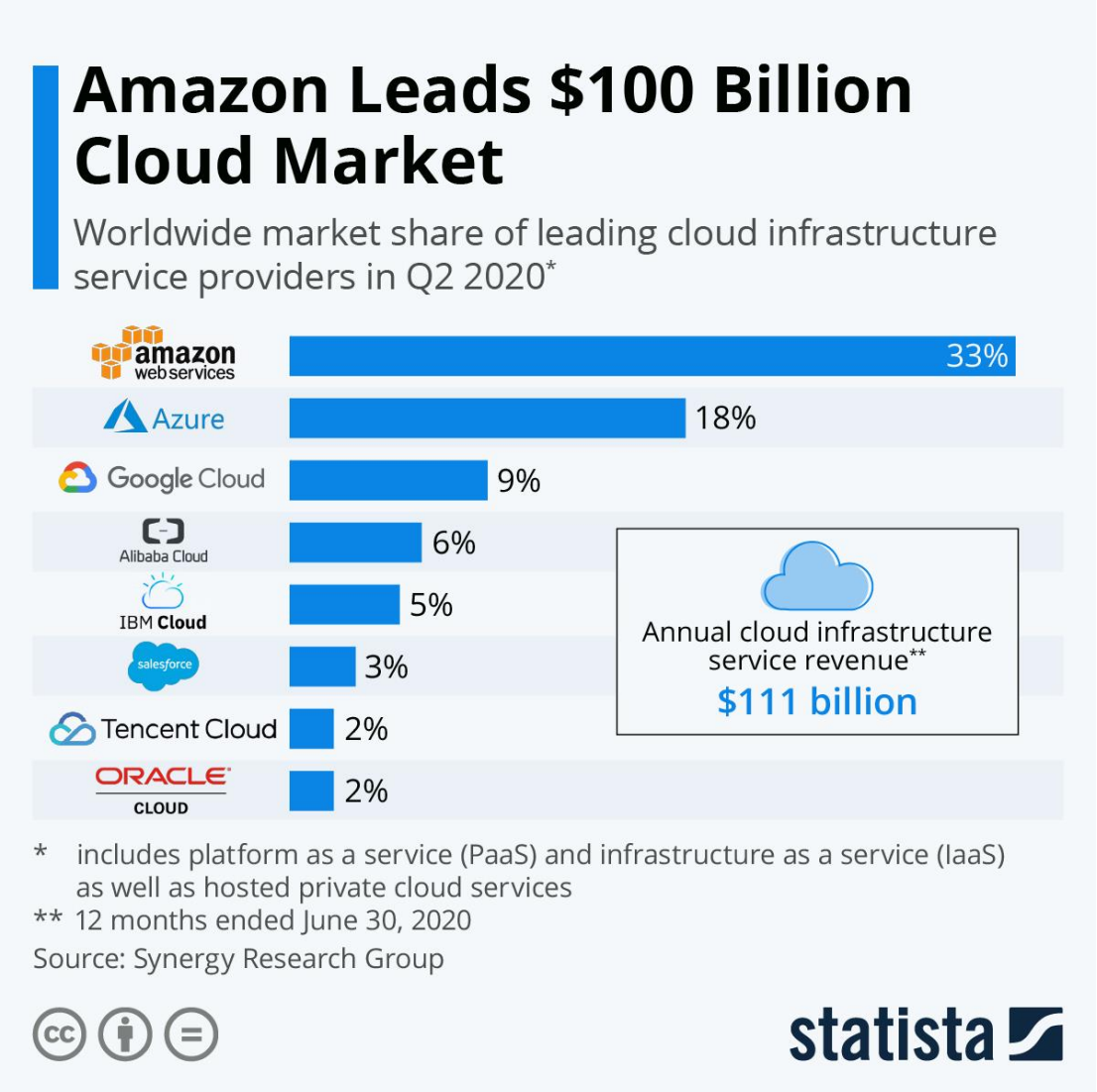

Źródło: https://www.smartnydom.pl/stacja-pogody-esp8266-afe-firmware/esp8266-wemos-d1-mini-pinout-2/

**Rys. 8. Płytka rozwojowa WeMos D1 mini wraz z opisem wyprowadzeń**

## 1.2. PRZETWARZANIE W CHMURZE

Technologia przetwarzania chmurowego (*chmura*) oferuje udostępnianie zasobów obliczeniowych oraz przestrzeni pamięciowej na żądanie przy pomocy Internetu. Rozwiązania oparte o chmurę charakteryzują się wysoką skalowalnością oraz niezawodnością[15]. Na rynku istnieje wiele dostawców rozwiązań chmurowych, chodź najpopularniejszymi i największymi rozwiązanymi są:

- Amazon Web Services (AWS)[16],
- Microsoft Azure[16],
- Google Cloud Platform[16].




Źródło: [16]

**Rys. 9. Udziały w światowym rynku wiodących dostawców usług chmurowych w drugim kwartale 2020 roku.**

### 1.2.1. Oferowane usługi

Sama infrastruktura chmury obliczeniowej to nic innego jak ogromne centra danych rozmieszczone w różnych zakątkach świata, zaopatrzone w wielkie ilości pamięci obliczeniowej i dyskowej. Owe zasoby nie miałby racji bytu gdyby nie usługi jakie w ich wykorzystaniem dostawcy mogą oferować. W świecie obliczeń w chmurze można wyróżnić trzy zasadnicze klasy usług

- Oprogramowanie jako usługa – *Software as a Service (SaaS)*

Dostawca udostępnia możliwość korzystania z oprogramowania uruchomionego na zdalnych serwerach (w chmurze)[17].

- Platforma jako usługa – *Platform as a Service* (PaaS)

Platforma w postaci udostępnianej przez dostawcę usługi to kompletne środowisko zawierające wszystkie komponenty potrzebne do udostępniania aplikacji bądź pracy programisty. Takie rozwiązanie wypiera tzw. podejście *On Premises* zakładające że wymagane oprogramowanie instalowane jest na fizycznej infrastrukturze klienta[17].

- Infrastruktura jako usługa – *Infrastructure as a Service* (IaaS)

W przypadku usług IaaS mowa tu o udostępnianiu przez dostawcę całych serwerów, sieci, pamięci masowych do użytku klienta[17].

Większość usług chmurowych charakteryzuje się rozliczaniem za ich wykorzystanie w konwencji *pay-as-you-go* oznacza to, że klient korzystający z usługi płaci tyle ile zasobów danej usługi wykorzystał, dobrym przykładem jest baza danych do której kierowane są zapytania. Klient zapłaci kwotę zależną od ilości zapytań skierowanych do bazy danych. Takie podejście pozwala na proste skalowanie i zarządzanie kosztami.

### 1.2.2. IoT w chmurze

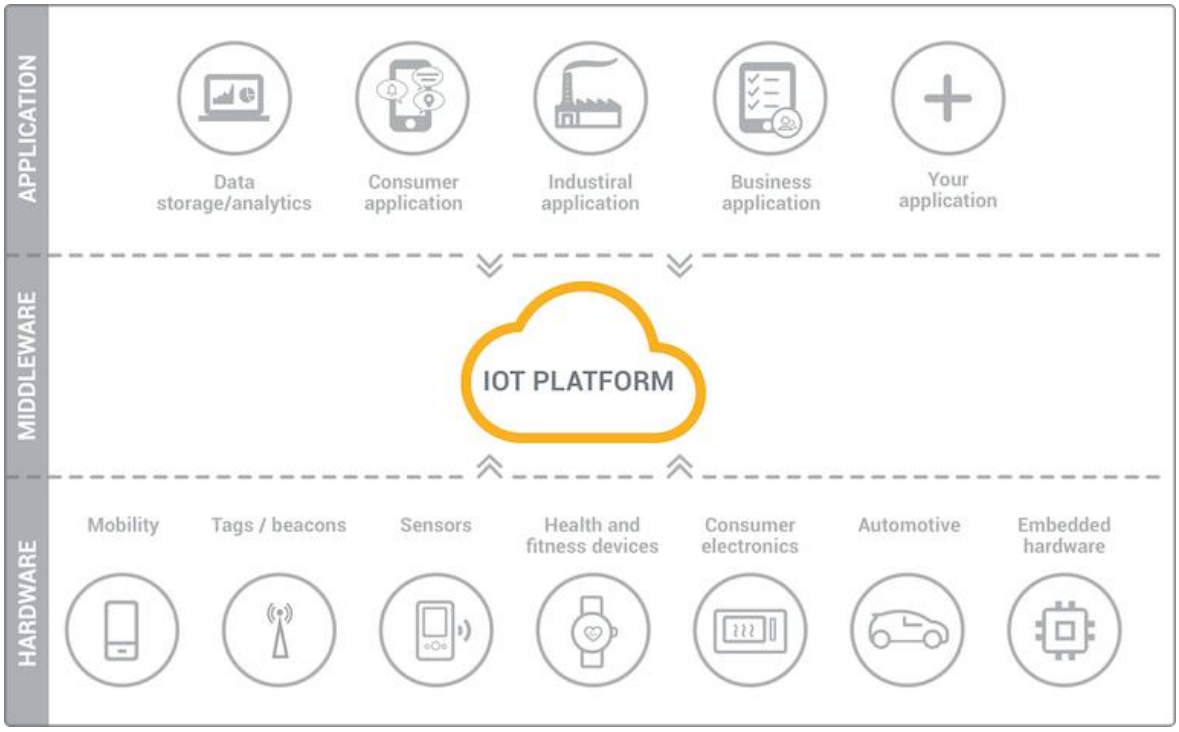
Można powiedzieć, że technologie chmurowe niemal idealnie wpasowują się w świat Internetu Rzeczy. Przeniesienie takich odpowiedzialności jak gromadzenie i analiza danych do chmury daje możliwości budowy systemu w oparciu o urządzenia z mocno ograniczoną mocą obliczeniową i pamięcią danych. Wykorzystanie usług

chmurowych pozwala także na budowę heterogenicznego i hermetycznego środowiska zarządzanego właśnie z poziomu chmury – wystarczy by urządzenia końcowe potrafiły prawidłowo komunikować się z usługą w chmurze.

Dostawcy rozwiązań chmurowych oferują kompletne zestawy usług przeznaczone stricte dla rozwiązań IoT – umożliwiają one nawiązanie połączenia z chmurą, przetwarzanie, gromadzenie i analizę danych a często też wiele więcej. Największe platformy IoT to:

- Azure IoT Suite (Microsoft),
- AWS IoT (Amazon),
- Google Cloud IoT.

Platformy IoT należy postrzegać jako warstwa pośrednicząca między urządzeniami Internetu Rzeczy a aplikacjami klienckimi. Usługa powinna umożliwić zarządzanie, utrzymanie i automatyzację pracy podłączonych urządzeń. Deweloperom platforma powinna zapewnić możliwość wykorzystania gotowych rozwiązań, przyspieszających proces tworzenia rozwiązania, a także łatwe utrzymanie oraz zapewnienie bezpieczeństwa[18].




Źródło: https://www.kaaproject.org/blog/what-is-iot-platform

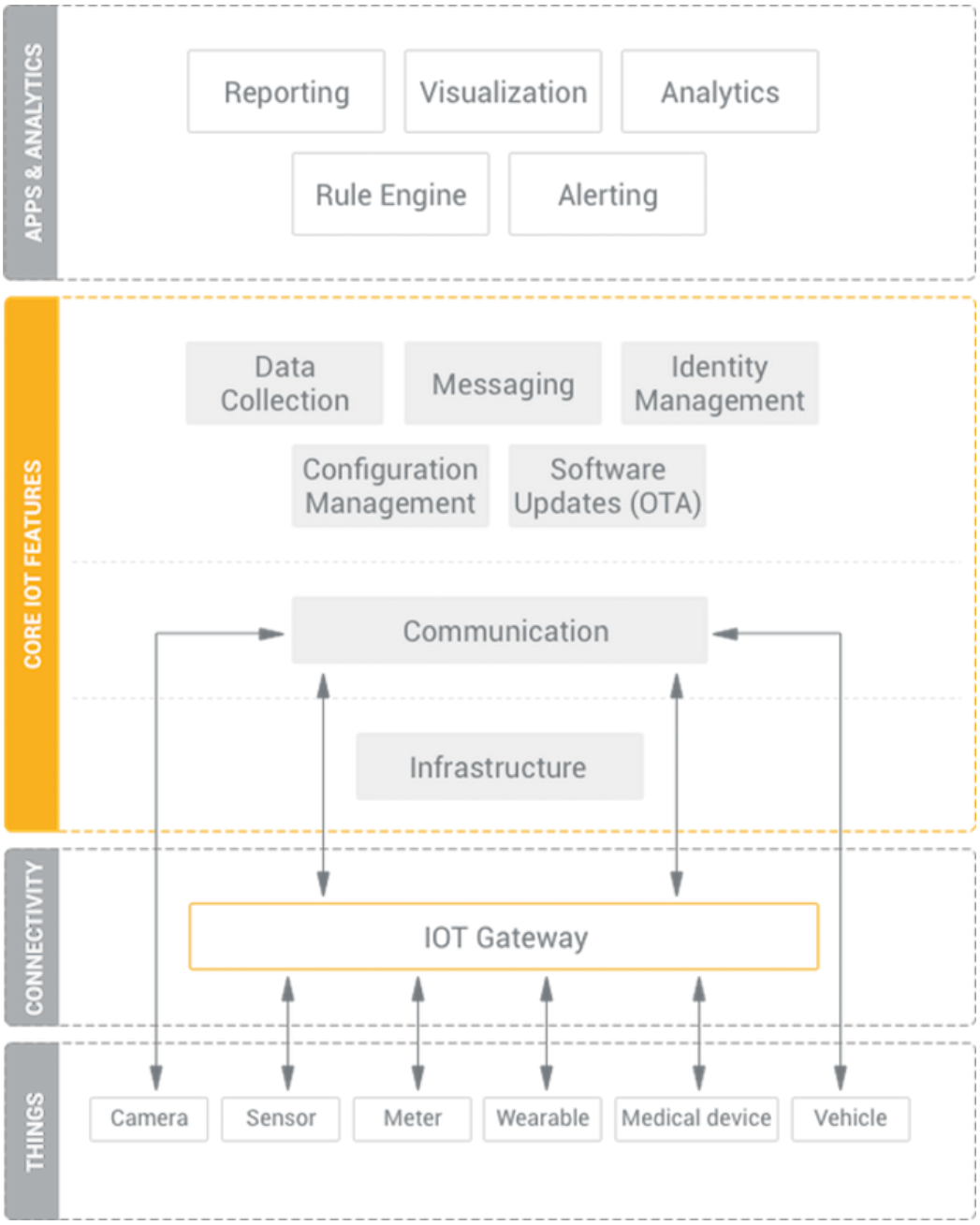
**Rys. 10. Platforma IoT jako warstwa pośrednicząca (ang. middleware)**

### 1.2.3. Komunikacja z chmurą

W rozwiązaniach Internetu Rzeczy komunikacja między urządzeniami IoT a chmurą może być organizowana w dwojaki sposób. Urządzenie może nawiązywać połączenie bezpośrednio z chmurą lub z pomocą tzw. Bramy (ang. Gateway) której często zadaniem jest konwersja protokołu komunikacyjnego bądź nawet pewne operacje obliczeniowe – takie podejście zostało nazwane przetwarzaniem na krawędzi sieci. Za przykład może posłużyć sytuacja kiedy urządzenia pracują we własnej sieci np. z wykorzystaniem systemu komunikacji LoRaWAN (ang. Long Range Wide Area Network[19]) ale zaistnieje potrzeba komunikacji sieci z chmurą – wtedy przy pomocą Bramy sieć może zostać ona skomunikowana z chmurą korzystającą z innego protokołu np. MQTT[18].

### 1.2.4. Stos technologiczny platformy IoT

Moduł podstawowych funkcjonalności chmurowej platformy IoT oparty jest o wiele elementów. Na samym dole znajduje się warstwa Infrastruktury, która umożliwia komunikację między różnymi komponentami chmury oraz orkiestracje zadań. Kolejna warstwa odpowiada za zapewnienie i utrzymanie obustronnej komunikacji między urządzeniami IoT a platformą chmurową. Następnie można wyodrębnić dedykowane usługi pozwalające na gromadzenie danych, powiadamianie i utrzymanie urządzeń oraz zapewnienie aktualizacji. Na samym szczycie znajdują się komponenty odpowiedzialne za analizę, raportowanie i wizualizację danych[18]. Wizualizacja opisanego stosu została przedstawiona na rysunku 11.




Źródło: https://www.kaaproject.org/blog/what-is-iot-platform

**Rys. 11. Stos technologiczny chmurowych platform IoT**

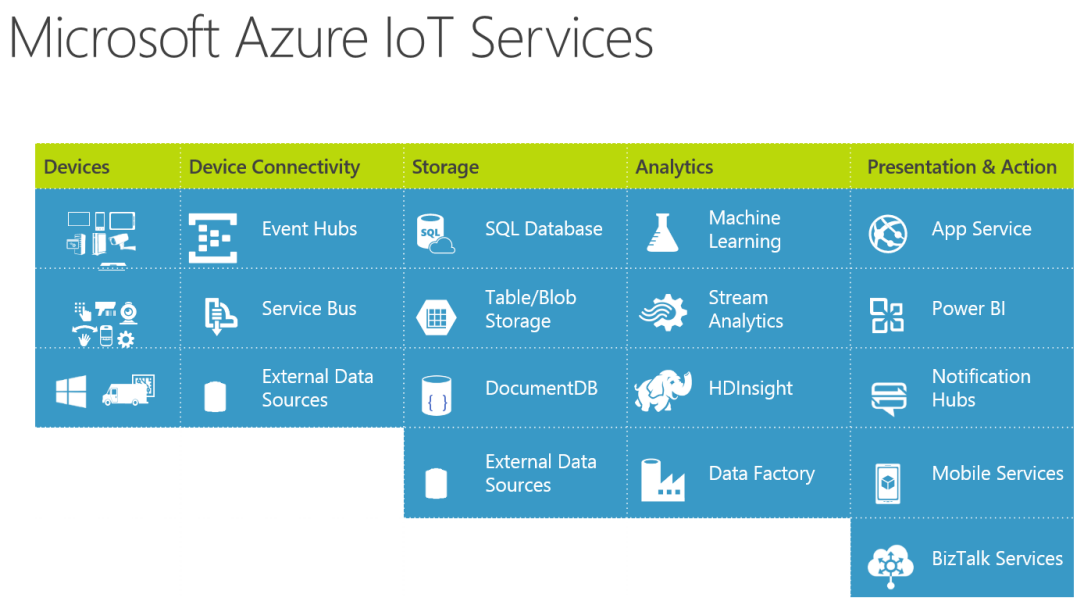
### 1.2.5. Przegląd dostępnych platform IoT

Każda z platform, oferowanych przez dostawców usług chmurowych, charakteryzuje się bardzo podobną ofertą serwisów dostępnych dla użytkownika. Dobór platformy często zależy od doświadczenia deweloperów w korzystaniu z danej technologii chmurowej a także od wielkości estymowanych kosztów.

- Azure IoT Suite (Microsoft)

Platforma oferowana przez czołowego producenta oprogramowania komputerowego charakteryzuje się dużym wsparciem we wstępnej fazie projektu, udostępnia w pełni funkcjonalne predefiniowane rozwiązania gotowe do pracy. Dodatkowym atutem jest opcja korzystania z wirtualnych urządzeń, co pozwala deweloperom na tworzenie prototypów i testowanie rozwiązań bez konieczności podłączania fizycznych urządzeń. Microsoft udostępnia także Azure IoT device SDK (Software Development Kit) dedykowany dla szeregu technologii i języków programowania, takich jak: ANSI C, Python, Node.js, .NET czy Java. To sprawia że budowanie heterogenicznych rozwiązań staje się możliwe a zarazem wygodne[20][21].

Protokoły komunikacji wspierane przez platforme Azure to HTTP, AMQP oraz MQTT.



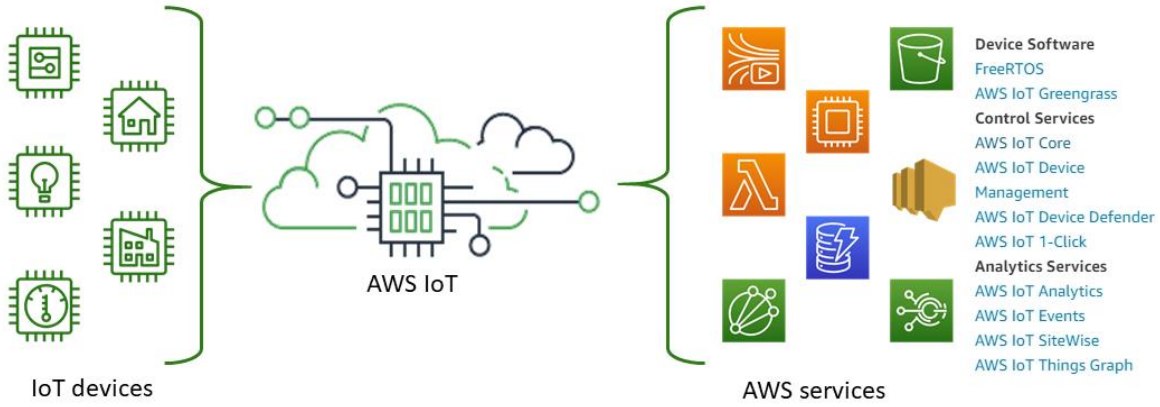

Źródło: https://bitstobrowser.wordpress.com/2015/10/24/connecting-a-device-to-azure-iot-suite/

**Rys. 12. Usługi dostępne w ramach platformy Azure IoT Suite**

- AWS IoT (Amazon)

Amazon podobnie jak Microsoft oferuje SDK natomiast wśród wspieranych technologii nie znajduje się .NET ale za to wspierane są rozwiązania oparte o iOS (firmy Apple)[22].

Komunikacja urządzeń z chmurą może odbywać się przy wykorzystaniu protokołów MQTT oraz HTTP 1.1. Amazon deklaruje bezpieczeństwo danych i obustronne uwierzytelnianie w każdym punkcie połączenia, uwierzytelnianie odbywa się za pomocą autorskiej metody *SigV4* opartej o certyfikacje. Deweloper może konfigurować certyfikaty oraz polityki dla podłączonych urządzeń, pozwala to np. na natychmiastowe odcięcie dostępu do chmury wybranemu urządzeniu – jeśli zaistnieje taka potrzeba[22].



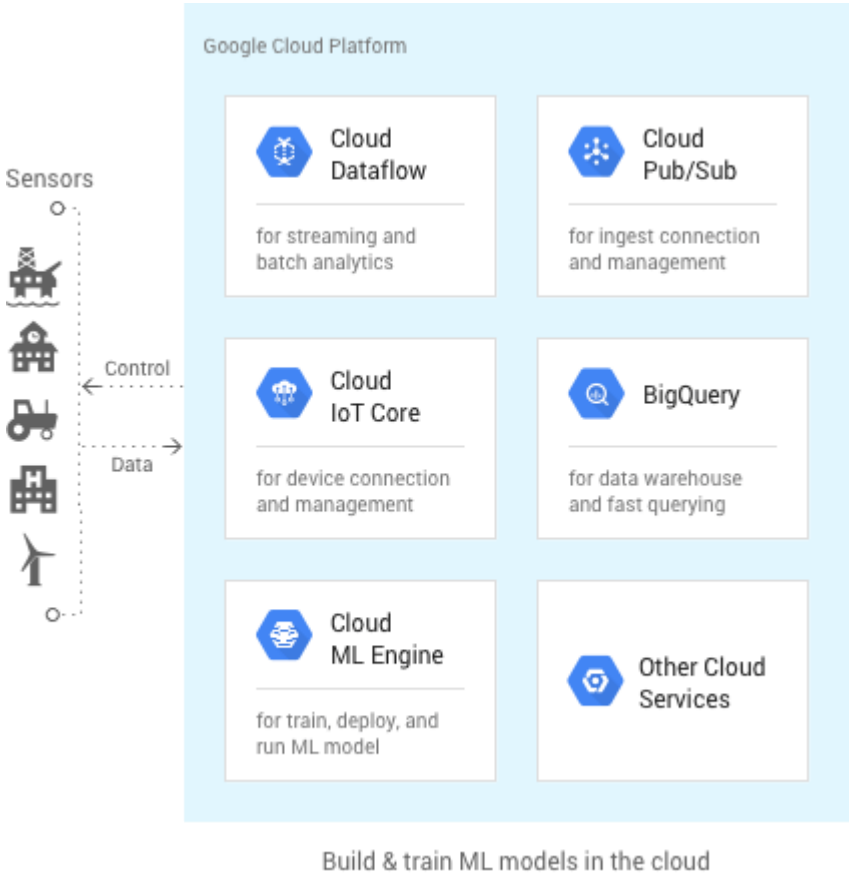

Źródło: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

**Rys. 13. Usługi dostępne w ramach platformy AWS IoT**

- Google Cloud IoT

W przeciwieństwie do poprzedników, Google udostępnia SDK jedynie dla języka C, natomiast dzięki udostępnianemu REST API społeczność skupiona wokół chmury Google utworzyła wiele projektów open-source umożliwiających prostą współprace z innymi technologiami (takimi jak: .NET czy Java).

Metody wymiany danych wpierane przez Google to MQTT oraz HTTP. Bezpieczeństwo komunikacji zapewniane jest dzięki protokołowi szyfrowania TLS 1.2[23].




Źródło: [23]

**Rys. 14. Usługi dostępne w ramach platformy Google Cloud IoT**

Zaprezentowane platformy chmurowe są do siebie bardzo podobne, a różnią się zaledwie szczegółami. Biorąc pod uwagę wszystkie czynniki, to Microsoft Azure IoT Suite zasługuje na miano najbardziej wszechstronnej, za sprawą tak wielu wspieranych języków programowania i technologii, największej ilości wspieranych protokołów wymiany informacji oraz wsparciu w początkowej fazie projektu dzięki predefiniowanym rozwiązaniom.

# ROZDZIAŁ II.

# PROJEKT SYSTEMU GROMADZENIA I ANALIZY DANYCH SENSORYCZNYCH Z URZĄDZEŃ IOT

## 2.1. Opis systemu

Projektowany system, ma pozwolić użytkownikom na łatwe i bezpieczne integrowanie własnych urządzeń IoT wyposażonych w sensory i podłączonych do sieci Internet. Każdy użytkownik powinien posiadać możliwość zarzadzania stanem pracującego w systemie urządzenia. Dodatkowo system ma pozwolić na przeglądanie zgromadzonych danych w określonym przedziale czasowym a także udostępnić użytkownikowi podstawowe analizy zgromadzonych informacji.

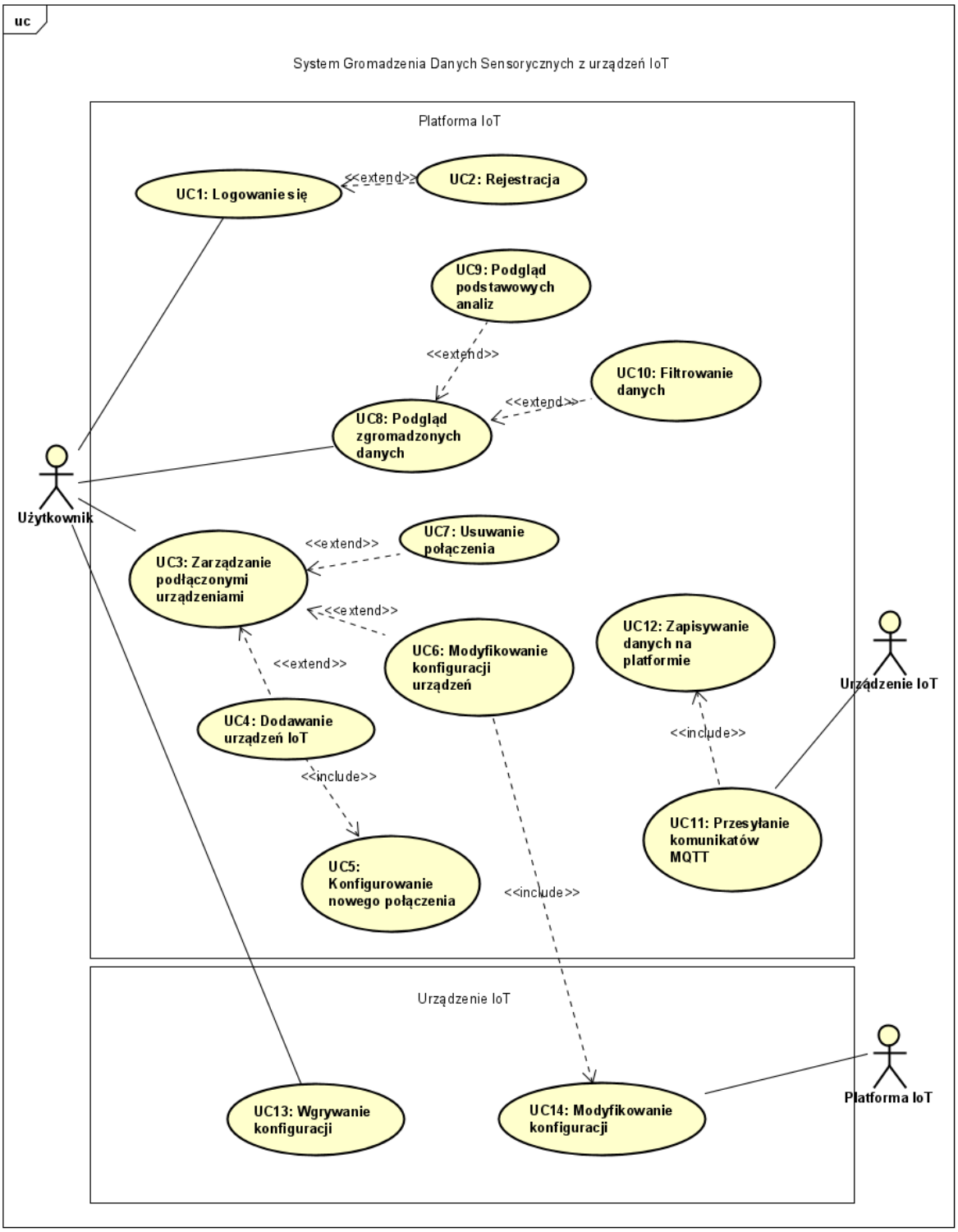
## 2.2. Wymagania

W celu specyfikacji wymagań posłużą: diagram przypadków użycia wraz z opisami przypadków – dla wymagań funkcjonalnych, oraz tabela – dla wymagań niefunkcjonalnych.

### 2.2.1. Funkcjonalne

Definicja aktorów w systemie:

- Użytkownik – użytkownik wchodzący bezpośrednio w interakcje w systemem, będący odbiorcą końcowym,
- Urządzenie IoT – urządzenie wyposażone w sensory, wykonujące pomiary wielkości fizycznych,
- Platforma IoT – System informatyczny funkcjonujący w oparciu o technologie chmurowe (zestaw serwisów chmurowych).




Źródło: Opracowanie własne przy pomocy narzędzia typu CASE *Astah UML*

**Rys. 15. Diagram przypadków użycia**

Przypadki użycia przedstawione na diagramie zostały zorganizowane w pakiety. Wprowadzenie takiej separacji pozwala lepiej odzwierciedlić w jakiej przestrzeni wykonywany jest określony przypadek użycia. Wszystkie przypadki umieszczone w pakiecie *Platforma IoT* oddziałują bezpośrednio na sam system

funkcjonujący w chmurze, natomiast te w pakiecie *Urządzenie IoT* działają wyłącznie na urządzenie. Ze względu na to, że może zachodzić interakcja między Platformą i Urządzeniem do diagramu zostali również wprowadzeni aktorzy o takich samych nazwach. Poniżej został zaprezentowany szczegółowy opis każdego z przypadków użycia.

**Tab. 2. Opis przypadku użycia UC1**

| | |
|---|---|
| Identyfikator przypadku użycia | UC1 |
| Nazwa przypadku użycia | Logowanie |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Użytkownik ma dostęp do aplikacji klienckiej |
| Scenariusz główny | 1) Użytkownik wybiera opcje zalogowania,<br>2) Użytkownik wypełnia formularz logowania,<br>3) Użytkownik zatwierdza formularz logowania. |
| Scenariusz alternatywny | 1) Użytkownik wybiera opcje zalogowania,<br>2) Użytkownik rozpoczyna procedurę zakładania nowego konta – następuje uruchomienie przypadku UC2. |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Użytkownik jest zalogowany |
| Zawarte i rozszerzone przypadki użycia | UC2 |

**Tab. 3. Opis przypadku użycia UC2**

| | |
|---|---|
| Identyfikator przypadku użycia | UC2 |
| Nazwa przypadku użycia | Rejestracja |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Użytkownik posiada dostęp do aplikacji klienckiej, brak konta użytkownika |
| Scenariusz główny | 1) Użytkownik wybiera opcje rejestracji nowego konta,<br>2) Użytkownik podaje adres e-mail, hasło oraz nazwę użytkownika,<br>3) Weryfikuje adres e-mail podając kod który otrzymał w wiadomości e-mail,<br>4) Zatwierdza rejestrację. |
| Scenariusz alternatywny | 1) Użytkownik wybiera opcje rejestracji nowego konta,<br>2) Użytkownik anuluje proces tworzenia nowego konta |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Zostaje utworzone konto użytkownika, następuje zalogowanie |
| Zawarte i rozszerzone przypadki użycia | Brak |

**Tab. 4. Opis przypadku użycia UC3**

| | |
|---|---|
| Identyfikator przypadku użycia | UC3 |
| Nazwa przypadku użycia | Zarządzanie podłączonymi urządzeniami |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Użytkownik jest zalogowany |
| Scenariusz główny | 1) Użytkownik wybiera opcję zarządzania urządzeniami,<br>2.1) Jeżeli użytkownik wybierze opcję dodawania urządzenia, zostaje uruchomiony przypadek UC4,<br>2.2) Jeżeli użytkownik wybierze opcję modyfikacji konfiguracji, zostaje uruchomiony przypadek UC6,<br>2.3) Jeżeli użytkownik wybierze opcję usunięcia połączenia, zostaje uruchomiony przypadek UC7. |
| Scenariusz alternatywny | Brak |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Użytkownik otrzymuje dostęp do opcji dodania nowego urządzenia oraz konfiguracji i usunięcia istniejących urządzeń (o ile wcześniej jakieś zostały dodane) |
| Zawarte i rozszerzone przypadki użycia | UC4, UC6, UC7 |

**Tab. 5. Opis przypadku użycia UC4**

| | |
|---|---|
| Identyfikator przypadku użycia | UC4 |
| Nazwa przypadku użycia | Dodawanie urządzeń IoT |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Zajście przypadku użycia UC3 |
| Scenariusz główny | 1) Użytkownik wypełnia formularz dodawania nowego urządzenia,<br>2) Wprowadza ID nowego urządzenie, nazwę, lokalizację urządzenia, określa wielkości fizyczne jakie mają być mierzone przez urządzenie a także interwał czasowy przesyłania wiadomości,<br>3) Zatwierdza formularz. |
| Scenariusz alternatywny | 1) Użytkownik rezygnuje z wypełniania formularza |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Następuje dodanie nowego urządzenia do zestawu urządzeń użytkownika |
| Zawarte i rozszerzone przypadki użycia | UC5 |

**Tab. 6. Opis przypadku użycia UC5**

| | |
|---|---|
| Identyfikator przypadku użycia | UC5 |
| Nazwa przypadku użycia | Konfigurowanie nowego połączenia |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Zajście przypadku użycia UC4 |
| Scenariusz główny | 1) Użytkownik ma możliwość pobrania pliku konfiguracyjnego dla dodanego urządzenia. |
| Scenariusz alternatywny | Brak |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Użytkownik zapisuje na dysku komputera plik konfiguracyjny dla dodanego urządzenia. |
| Zawarte i rozszerzone przypadki użycia | Brak |

**Tab. 7. Opis przypadku użycia UC6**

| | |
|---|---|
| Identyfikator przypadku użycia | UC6 |
| Nazwa przypadku użycia | Modyfikowanie konfiguracji urządzeń |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Zajście przypadku użycia UC3 |
| Scenariusz główny | 1) Użytkownik wybiera urządzenie z zestawu swoich urządzeń,<br>2) W formularzu edytuje konfiguracje urządzenia,<br>3) Zatwierdza edytowany formularz. |
| Scenariusz alternatywny | 1) Użytkownik rezygnuje z edycji konfiguracji urządzenia. |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Następuje edycja konfiguracji wybranego urządzenia |
| Zawarte i rozszerzone przypadki użycia | UC14 |

**Tab. 8. Opis przypadku użycia UC7**

| | |
|---|---|
| Identyfikator przypadku użycia | UC7 |
| Nazwa przypadku użycia | Usuwanie połączenia |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Zajście przypadku użycia UC3 |
| Scenariusz główny | 1) Użytkownik wybiera urządzenie do usunięcia,<br>2) Zatwierdza usunięcia połączenia. |
| Scenariusz alternatywny | 1) Użytkownik rezygnuje z usuwania wybranego urządzenia. |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Następuje usunięcie połączenia wybranego urządzenia, użytkownik nie posiada już usuniętego urządzenia w zestawie swoich urządzeń |
| Zawarte i rozszerzone przypadki użycia | Brak |

**Tab. 9. Opis przypadku użycia UC8**

| | |
|---|---|
| Identyfikator przypadku użycia | UC8 |
| Nazwa przypadku użycia | Podgląd zgromadzonych danych |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Użytkownik jest zalogowany |
| Scenariusz główny | 1) Użytkownik wybiera opcje podglądu zgromadzonych danych,<br>2) Wybiera nazwę urządzenia z którego dane chce zobaczyć,<br>3) Jeżeli użytkownik wybierze opcje podglądu analiz, zostaje uruchomiony przypadek UC9,<br>3.1) Jeżeli użytkownik wybierze opcje filtrowania danych, zostaje uruchomiony przypadek UC10. |
| Scenariusz alternatywny | 1) Wyświetlenie komunikatu o braku danych z wybranego urządzenia |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Wyświetlenie zgormadzonych danych w postaci tabeli |
| Zawarte i rozszerzone przypadki użycia | UC9, UC10 |

**Tab. 10. Opis przypadku użycia UC9**

| | |
|---|---|
| Identyfikator przypadku użycia | UC9 |
| Nazwa przypadku użycia | Podgląd podstawowych analiz |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Zajście przypadku użycia UC8 |
| Scenariusz główny | 1) Użytkownik wybiera opcję wyświetlenia podstawowych analiz zebranych danych |
| Scenariusz alternatywny | 1) W przypadku braku danych, użytkownikowi ukazuje się komunikat o braku danych. |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Wyświetlenie przebiegów czasowych zgromadzonych wartości, wartości średnich oraz minimalnych i maksymalnych. |
| Zawarte i rozszerzone przypadki użycia | Brak |

**Tab. 11. Opis przypadku użycia UC10**

| | |
|---|---|
| Identyfikator przypadku użycia | UC10 |
| Nazwa przypadku użycia | Filtrowanie danych |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Zajście przypadku użycia UC8 |
| Scenariusz główny | 1) Użytkownik wybiera opcje filtrowania danych,<br>2) Wprowadza datę początkową i końcową okresu czasu z którego analizy mają zostać przeprowadzone, |
| Scenariusz alternatywny | 1) Wyświetlanie analiz dla wszystkich zgromadzonych danych |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Zostaje zawężony zestaw analizowanych danych do danych z okresu zdefiniowanego w filtrach. |
| Zawarte i rozszerzone przypadki użycia | Brak |

**Tab. 12. Opis przypadku użycia UC11**

| | |
|---|---|
| Identyfikator przypadku użycia | UC11 |
| Nazwa przypadku użycia | Przesyłanie komunikatów MQTT |
| Akor inicjalizujący | Urządzenie IoT |
| Warunki wstępne | Połączenie między urządzeniem a platformą IoT, włączone wysyłanie danych z urządzenia. |
| Scenariusz główny | 1) Urządzenie odczytuje dane z sensorów,<br>2) Urządzenie przesyła wiadomość do platformy . |
| Scenariusz alternatywny | 1) Gdy wyłączone jest gromadzenie danych z danego urządzenia, urządzenie odczytuje dane z sensorów lecz nie przesyła ich do platformy. |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Wiadomość jest dostarczana do platformy |
| Zawarte i rozszerzone przypadki użycia | UC12 |

**Tab. 13. Opis przypadku użycia UC12**

| | |
|---|---|
| Identyfikator przypadku użycia | UC12 |
| Nazwa przypadku użycia | Zapisanie danych na platformie |
| Akor inicjalizujący | Urządzenie IoT |
| Warunki wstępne | Zajście przypadku użycia UC11 |
| Scenariusz główny | 1) Wiadomość dociera do platformy, |
| Scenariusz alternatywny | 1) Wiadomość nie dociera do platformy |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Wiadomość zostaje zapisana w bazie danych |
| Zawarte i rozszerzone przypadki użycia | Brak |

**Tab. 14. Opis przypadku użycia UC13**

| | |
|---|---|
| Identyfikator przypadku użycia | UC13 |
| Nazwa przypadku użycia | Wgranie konfiguracji |
| Akor inicjalizujący | Użytkownik |
| Warunki wstępne | Urządzenie zostało wcześniej skonfigurowane a użytkownik posiada plik konfiguracyjny na dysku. |
| Scenariusz główny | 1) Użytkownik wgrywa plik konfiguracyjny do pamięci urządzenia,<br>2) Użytkownik ponownie uruchamia urządzenie |
| Scenariusz alternatywny | Brak |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Przy ponownym uruchamianiu urządzenie dokonuje automatycznego nawiązania połączenia z platformą |
| Zawarte i rozszerzone przypadki użycia | Brak |

**Tab. 15. Opis przypadku użycia UC14**

| | |
|---|---|
| Identyfikator przypadku użycia | UC14 |
| Nazwa przypadku użycia | Modyfikowanie konfiguracji |
| Akor inicjalizujący | Urządzenie IoT |
| Warunki wstępne | Zmodyfikowanie konfiguracji wybranego urządzenia przez użytkownika w aplikacji klienckiej (zajście przypadku użycia UC6). |
| Scenariusz główny | 1) Urządzenie otrzymuje komunikat o modyfikacji konfiguracji przez użytkownika,<br>2) Urządzenie przeprowadza modyfikacje konfiguracji. |
| Scenariusz alternatywny | Brak |
| Po spełnieniu warunków i wykonaniu scenariusz głównego | Urządzenie kontynuuje prace w oparciu o nową konfiguracje |
| Zawarte i rozszerzone przypadki użycia | Brak |

### 2.2.2. Pozafunkcjonalne

**Tab. 16. Specyfikacja wymagań pozafunkcjonalnych**

| L.p. | Opis |
|---|---|
| 1. | Połączenie urządzeń z platformą oraz ich uwierzytelnianie będzie następowało przy pomocy kluczy symetrycznych. |
| 2. | Modyfikacja konfiguracji po stronie pracującego urządzenia powinna trwać nie dłużej niż 60 s. |
| 3. | Aplikacja kliencka powinna być możliwa do uruchomienia przez takie przeglądarki internetowe jak Google Chrome (w wersji > 88.0), Opera (w wersji > 73.0) i Mozilla Firefox (w wersji > 84.0). |
| 4. | Wykorzystane komponenty chmurowe powinny charakteryzować dostępnością na poziomie 99,999% (w skali „dziewiątek” wysokiej dostępności[24]. |
| 5. | System w trybie gromadzenia danych powinien pozwolić na gromadzenie co najmniej 99% przesyłanych przez urządzenia wiadomości. |
| 6. | Opóźnienie między wysłaniem a otrzymaniem wiadomości powinno wynosić maksymalnie 1 s. |
| 7. | System powinien być w stanie obsłużyć, w sumie, co najmniej 7000 widomości na dobę. |

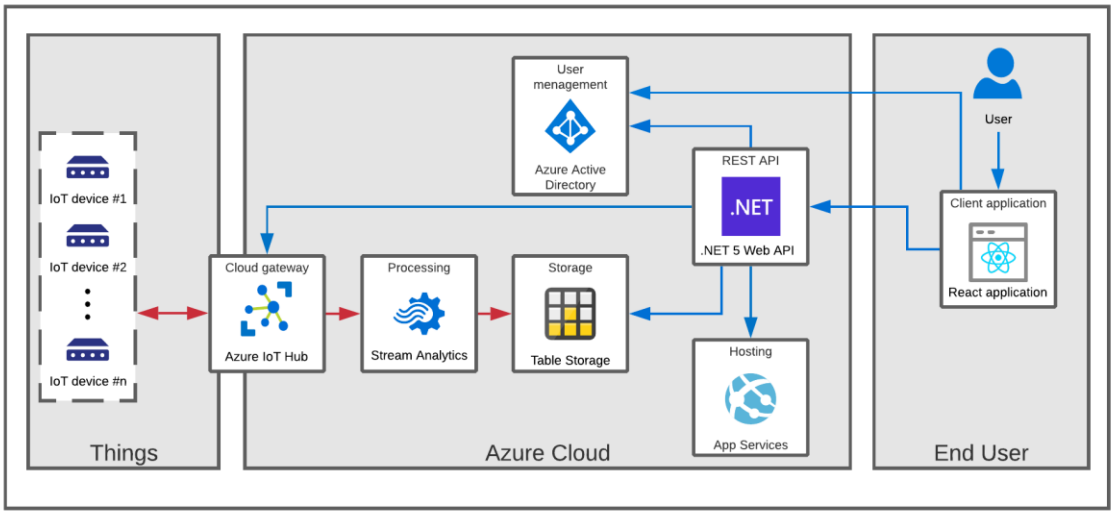
## 2.3. Architektura rozwiązania

### 2.3.1. Architektura wysokopoziomowa

Na Rysunku 16. został umieszczony diagram prezentujący wysokopoziomową architekturę projektowanego systemu. Na najwyższym poziomie abstrakcji wyróżnić można 3 zasadnicze elementy systemu:

- Urządzenia Internetu Rzeczy

Warstwa brzegowa odpowiadająca za przeprowadzanie pomiarów wielkości fizycznych i przesłanie danych do platformy – odpowiedzialna za akwizycję danych. Za urządzenia posłużą mikrokomputery Raspberry Pi opisane we wstępie teoretycznym pracy, ze względu na swoją uniwersalność oraz szeroki wachlarz dostępnych rozszerzeń.




Źródło: Opracowanie własne z wykorzystaniem narzędzia Lucidchart w wersji darmowej

**Rys. 16. Diagram wysokopoziomowej architektury Systemu gromadzenia i analizy danych sensorycznych z urządzeń IoT w wykorzystaniem przetwarzania w chmurze**

- Infrastruktura chmurowa

Na platformę zapewniającą serwisy chmurowe została wybrana chmura Microsoft Azure – jest jedną z wiodących na rynku rozwiązań chmurowych technologią, dodatkowo zapewnia dostępność serwisów na poziomie, który całkowicie spełnia stawiane projektowanemu systemowi wymagania. Ponadto jako jedyna z zaprezentowanych we wstępie teoretycznym całkowicie wspiera technologię .NET.

- Aplikacja kliencka

Za aplikacje kliencką posłuży aplikacja webowa – takie rozwiązanie jest najbardziej uniwersalne, ponieważ w przeciwieństwie do rozwiązań desktopowych oraz mobilnych nie jest uzależnione od platformy uruchomieniowej a tym samym daje dostęp szerokiej grupie użytkowników. Dodatkowo aplikacje webowe gwarantują łatwe utrzymanie, aktualizację, ogromną skalowalność i podatność na rozbudowę funkcjonalności.

W celu lepszego wyjaśniania powiązań między komponentami, na diagramie wysokopoziomowej architektury zamieszczone zostały strzałki w dwóch kolorach. Strzałki w kolorze czerwonym pokazują przepływ wiadomości z danymi –

telemetrycznymi, generowanymi przez urządzenia, oraz konfiguracyjnymi między chmurą a urządzeniami. Niebieskie strzałki wskazują na zależność między elementami systemu, tzn. że np. Aplikacja kliencka *Client application* zależy (bezpośrednio korzysta) od REST API.

### 2.3.2. Architektura szczegółowa

Postępując zgodnie z zasadą dekompozycji i schodząc poziom niżej można wyróżnić następujące serwisy i technologie zawarte we wcześniej przedstawionych elementach.

- Oprogramowanie i budowa urządzeń brzegowych

Mikrokomputery Raspberry Pi wraz z płytką rozszerzeniową Sense HAT wyposażoną w sensory pozwalające mierzyć: temperaturę i wilgotność powietrza, ciśnienie atmosferyczne, a także przyspieszenie, pole magnetyczne oraz orientację urządzenia w przestrzeni. Oprogramowanie uruchamiane na urządzeniach napisane w języku Python 3.7.

- Komunikacja warstwy urządzeń z chmurą

Do komunikacji wykorzystany zostanie protokół MQTT będący standardowym protokołem wykorzystywanym w rozwiązaniach IoT.

- Azure IoT Hub

Serwis platformy chmurowej Azure będący podstawą każdego rozwiązania IoT budowanego w oparciu o chmurę firmy Microsoft. Azure IoT Hub odpowiada za obustronną komunikację między infrastrukturą IoT w chmurze a urządzeniami brzegowymi. W komunikacji urządzenie-chmura odbiera wszystkie wiadomości przesyłane z urządzeń, dodatkowo pozwala na przesyłanie wiadomości i powiadomień na ścieżce chmura-urządzenie. W kontekście całej platformy jest swego rodzaju brokerem MQTT, niemiej nie wspiera on wszystkich funkcji wyróżnionych w specyfikacji standardu MQTT v3.1.1, co zostało podkreślone w dokumentacji. Dodatkowo przechowuje po stronie chmury odwzorowanie każdego urządzenia odzwierciedlające jego bieżącą konfigurację. Wszystkie urządzenia komunikujące się z IoT Hub'em muszą korzystać z protokołu szyfrowania warstwy transportu danych – TLS. Do komunikacji z tym serwisem mogą zostać wykorzystane biblioteki udostępniane przez

firmę Microsoft – Azure IoT SDKs lub bezpośrednio protokół MQTT. Biblioteki zapewniają pewną warstwę abstrakcji i są prostsze w wykorzystaniu dla programistów. Azure Python SDK dla urządzeń domyślnie wspiera protokół MQTT i dodatkowe konfigurację z poziomu kodu nie są wymagane[25][26].

- Azure Stream Analytics

Serwis pozwalający na przetwarzanie danych w czasie rzeczywistym. Przechwytuje on dane przychodzące do IoT Hub'a i przeprowadza na nich operacje. W budowanym rozwiązaniu odpowiada on za zapisywanie otrzymywanych danych w bazie danych.

- Table Storage

Przestrzenią na dane przesyłane przez urządzenia jest serwis Table Storage pozwalających na przechowywanie danych nierelacyjnych – NoSQL. Przechowywane dane nie posiadają z góry określonego modelu a budowa encji oparta jest o schemat klucz-wartość, co za tym idzie taka baza wpasowuje się idealnie w potrzeby rozwiązania IoT. Format przesyłanych wiadomości z różnych urządzeń może być odmienny lub na przestrzeni czasu może zajść potrzeba zmiany takiego formatu, dzięki przechowywaniu danych w postaci tabeli nie stanowi to żadnego problemu.

Wyżej opisane elementy są związane bezpośrednio z infrastrukturą IoT i gromadzeniem danych z urządzeń, kolejne elementy które zostaną zaprezentowane są związane z zarządzaniem pracą systemu oraz jego użytkownikami.

- Azure Web Services

Serwis oparty o protokół HTTP pozwalający na hosting aplikacji internetowych czy też Web API. W projektowanym rozwiązaniu odpowiada za publicznie udostępnianie aplikacji klienckiej. Oferuje on środowiska uruchomieniowe dla szeregu technologii, takich jak: .NET, .NET Core, Java, Node.js czy Python. Dzięki wykorzystaniu takiego rozwiązania, nie ma konieczności martwienia się o konfigurowanie serwera, portów czy choćby bezpieczeństwo samej aplikacji. Dodatkowo zapewnia on szerokie możliwości skalowania i balansowania obciążeniem.

- REST API

Web API z zachowaniem standardów REST udostępniające możliwość zarządzania zasobami w systemie, publikowane przez Azure Web Services, będzie tworzone w oparciu o technologię .NET 5.

- Aplikacja kliencka

Aplikacja webowa uruchamiana w przeglądarce użytkownika tworzona w konwencji SPA – (*Single Page Application* – aplikacja jednostronicowa). Aplikacja hostowana przez Web API jako jeden z punktów końcowych. Udostępnia ona interfejs użytkownika pozwalający na zarządzanie systemem. Aplikacja frontendowa zostanie zbudowana z wykorzystaniem języka TypeScript.

- Azure Active Directory

Serwis pozwalający na zarządzenie użytkownikami systemu, ich kontami, dostępem do zasobów a także udostępniający mechanizmy uwierzytelniania i autoryzacji użytkowników. Wykorzystanie takiego serwisu charakteryzuje się wysokim bezpieczeństwem w stosunku do rozwiązań budowanych od podstaw, w dodatku idealnie wpisuje się w rozwiązania chmurowe dzięki możliwości szybkiej adaptacji takiego serwisu pod własne potrzeby, a także łatwą integrację z innymi serwisami chmurowymi.

## 2.4. Projekt platformy

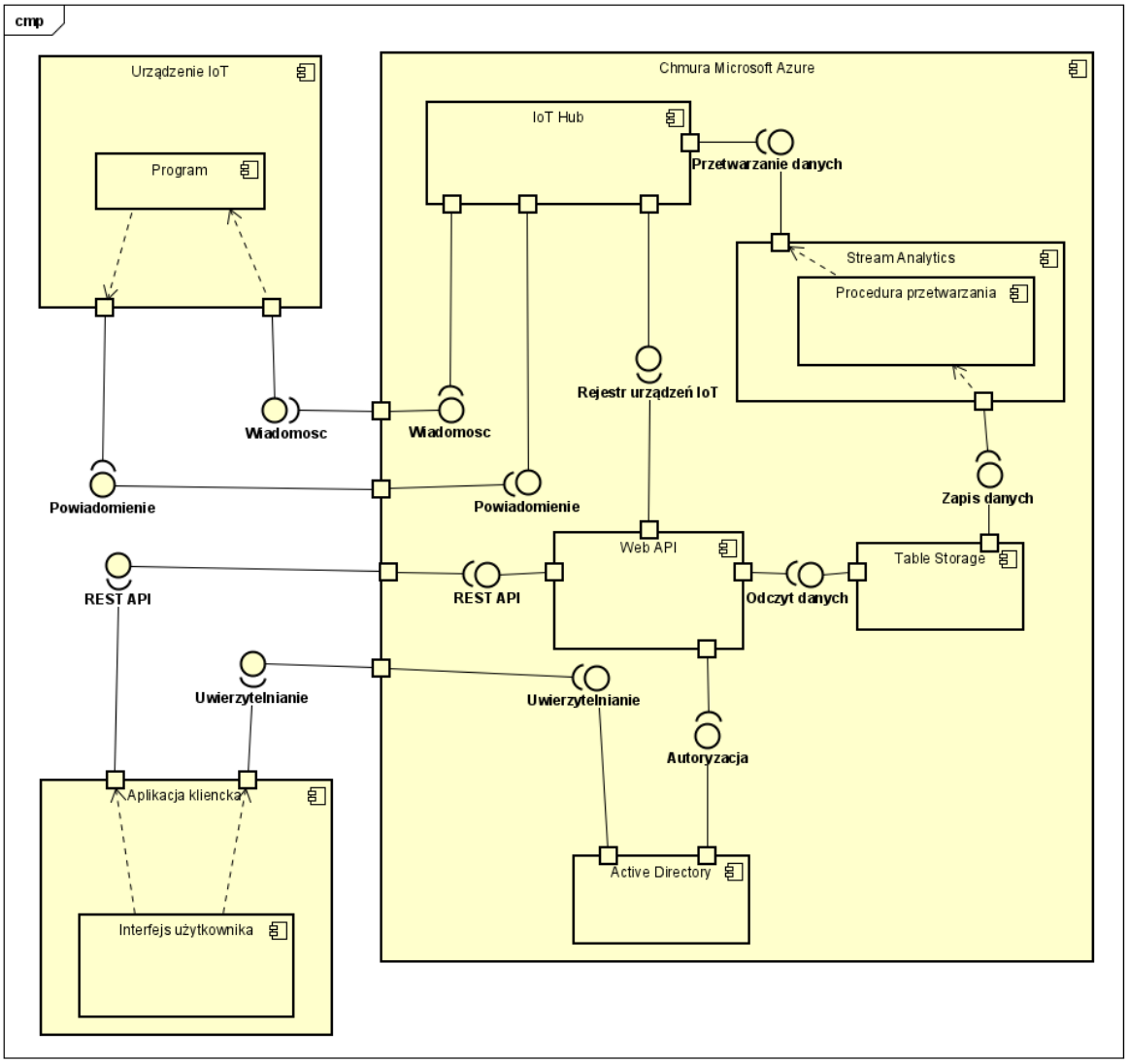
### 2.4.1. Komponenty systemu

Zgodnie z wcześniej przedstawioną architekturą rozwiązania, wykonałem diagram komponentów systemu.

Diagram został przedstawiony na rysunku 17. Zasadnicze komponenty odpowiadają trzem podstawowym elementom opisanym w punkcie 2.3.1. Dodatkowo dzięki przedstawieniu interfejsów dostarczanych i wymaganych przez poszczególne komponenty diagram lepiej oddaje interakcje między elementami systemu.

Urządzenie IoT wykonując program na nim uruchamiany dokonuje pomiaru wartości fizycznych i w postaci wiadomości udostępnia dane na zewnątrz co określoną jednostkę czasu. Dane udostępniane przez urządzenie są konsumowane przez

chmurę, a dokładniej przez komponent IoT Hub do którego dane z zewnątrz bezpośrednio docierają. Dodatkowo między tymi dwom komponentami istnieje mechanizm powiadomień, oznacza to że IoT Hub może przesłać powiadomienie do urządzenia. Komunikacja między urządzeniem a IoT Hub'em powinna przebiegać asynchronicznie, oznacza to że otrzymanie powiadomienia przez urządzenie nie powinno zaburzać procedury cyklicznego przesyłania wiadomości z odczytami z czujników.



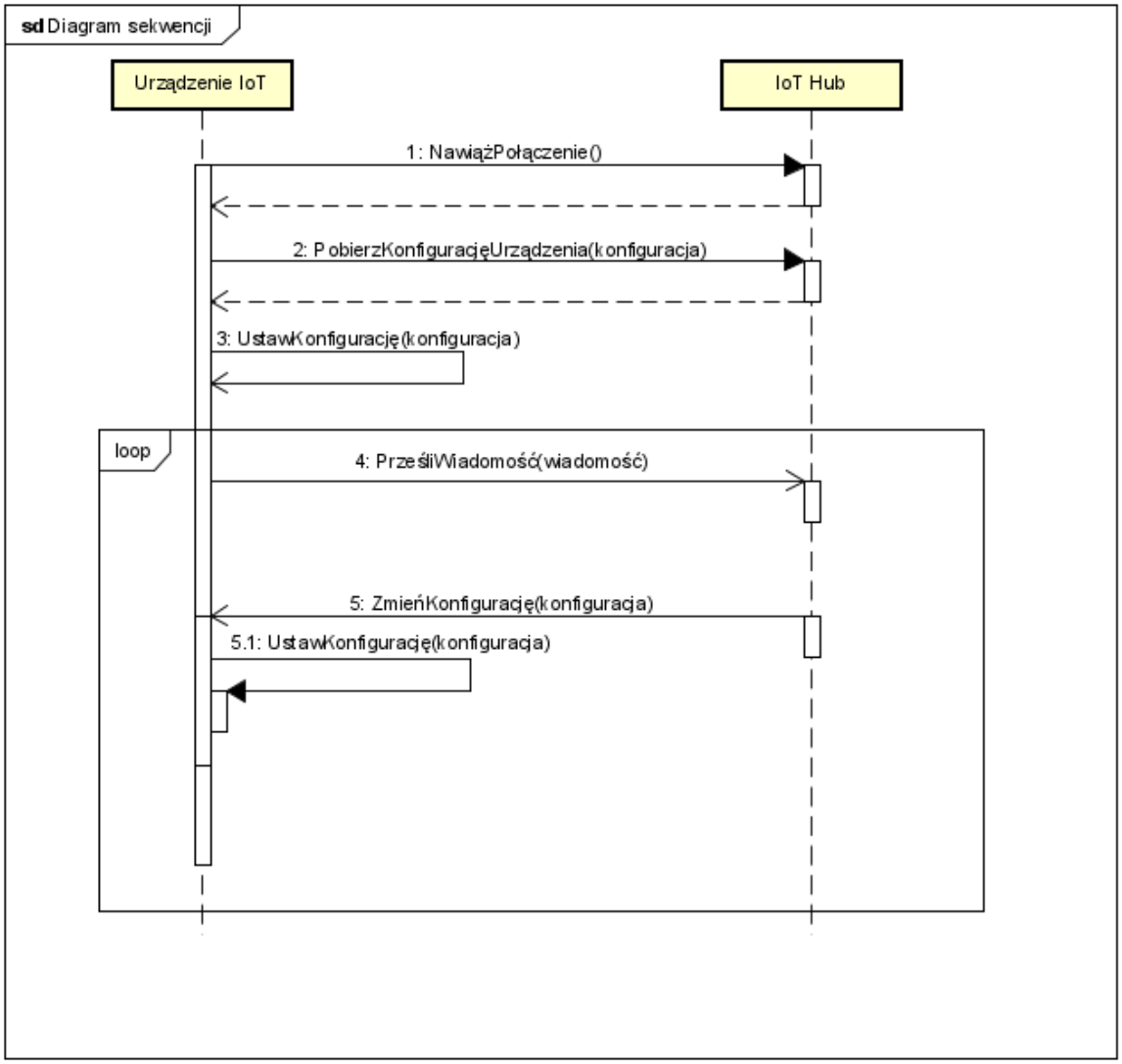

Źródło: Opracowanie własne przy pomocy narzędzia typu CASE: *Astah UML*

**Rys. 17. Diagram komponentów Systemu gromadzenia i analizy danych sensorycznych z urządzeń IoT w wykorzystaniem przetwarzania w chmurze**

Diagram sekwencji obrazujący jak owa komunikacja przebiega, został przedstawiony na rysunku 18. Komunikacja rozpoczyna się od nawiązania połączenia, następnie Urządzenie pobiera konfiguracje przechowywaną przez IoT Hub, po ustawieniu konfiguracji rozpoczyna się główna pętla w której asynchronicznie wykonywane

są dwa zadania – przesyłanie wiadomości oraz zmiana konfiguracji urządzenia w przypadku gdy urządzenie otrzyma powiadomienie o zmianie konfiguracji ze strony IoT Hub'a.

Na rysunku 19. został przedstawiony diagram klas programu uruchomianego na Urządzeniu IoT.

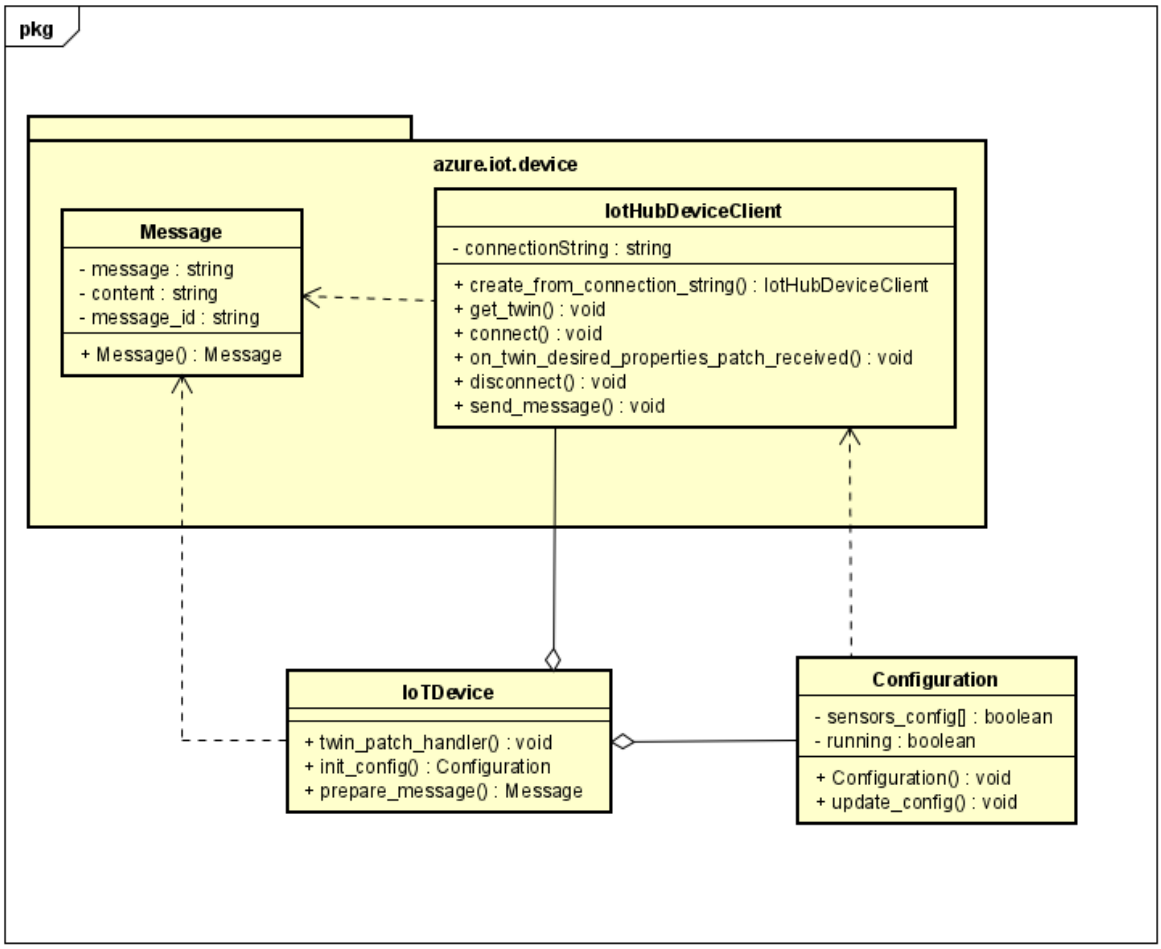



Źródło: Opracowanie własne przy pomocy narzędzia typu CASE: *Astah UML*

**Rys. 18. Diagram sekwencji komunikacji między komponentami Urządzenie IoT a IoT Hub**

W celu zrealizowania komunikacji między Urządzeniem IoT a IoT Hub'em Microsoft udostępnia zestaw klas *Azure Python SDK* dla urządzeń IoT. W zestawie znajdują się klasy z których program uruchamiany na urządzeniu może skorzystać. Klasą która udostępnia bezpośrednio możliwość nawiązania połączenia na podstawie tzw. *ConnectionString* (z języka angielskiego można tłumaczyć jako klucz

połączeniowy) i implementuje po stronie urządzenia obiekt klienta do obsługi serwisu IoT Hub jest *IoTHubDeviceClient.* Klient ten pozwala na nawiązanie połączenia z hubem – metoda *connect(),* pozyskanie konfiguracji urządzenia z IoT Hub'a – metoda *get_twin(),* zdefiniowanie funkcji obsługi powiadomienia o zmianie konfiguracji – metoda *on_twin_desired_properties_patch_received(),* oraz oczywiście przesyłanie wiadomości a także zakończenie połączenia.

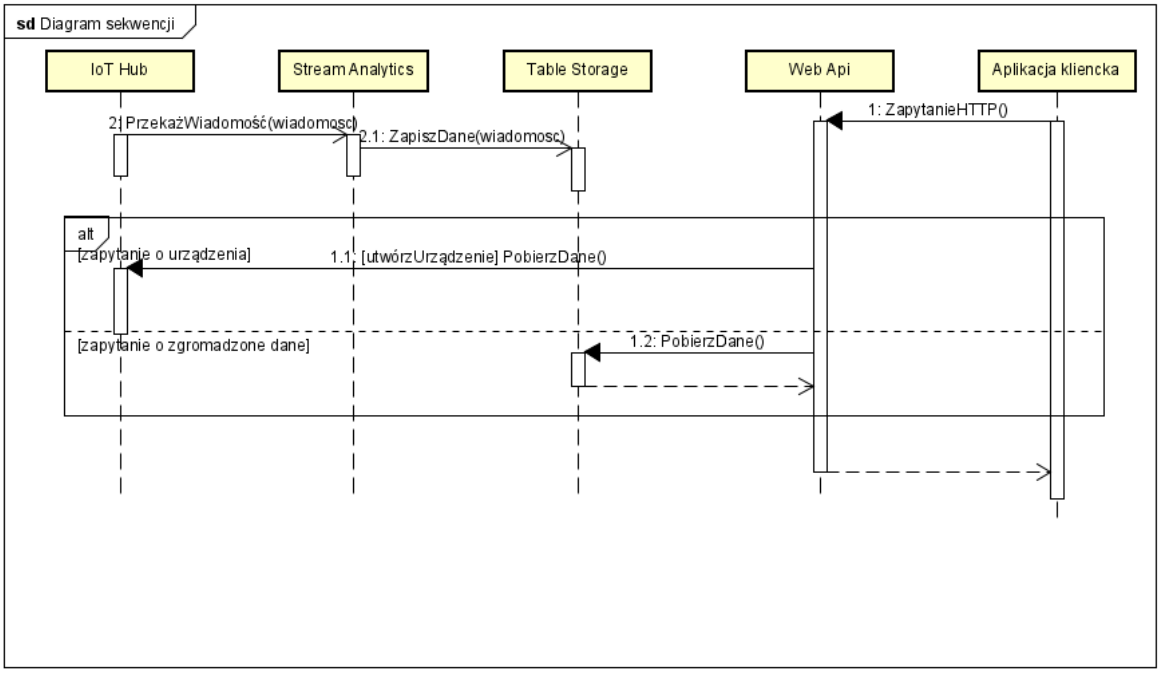



Źródło: Opracowanie własne przy pomocy narzędzia typu CASE: *Astah UML*

**Rys. 19. Diagram klas programu uruchomianego na Urządzeniu IoT**

IoT Hub może jedynie odbierać dane z Urządzeń IoT i w celu operowania na nich należy wykorzystać inny mechanizm udostępniający możliwość przetwarzania danych. W tym wypadku jest to komponent Stream Analytics, wykonuje on zadaną procedurę na danych docierających do IoT Hub'a w czasie rzeczywistym. W budowanym rozwiązaniu jest on pośrednikiem pomiędzy IoT Hub'em, czyli węzłem do którego docierają wiadomości ze wszystkich urządzeń, a bazą danych. Baza danych poza przechowywaniem danych udostępnia możliwość zapisu jak i również odczytu.

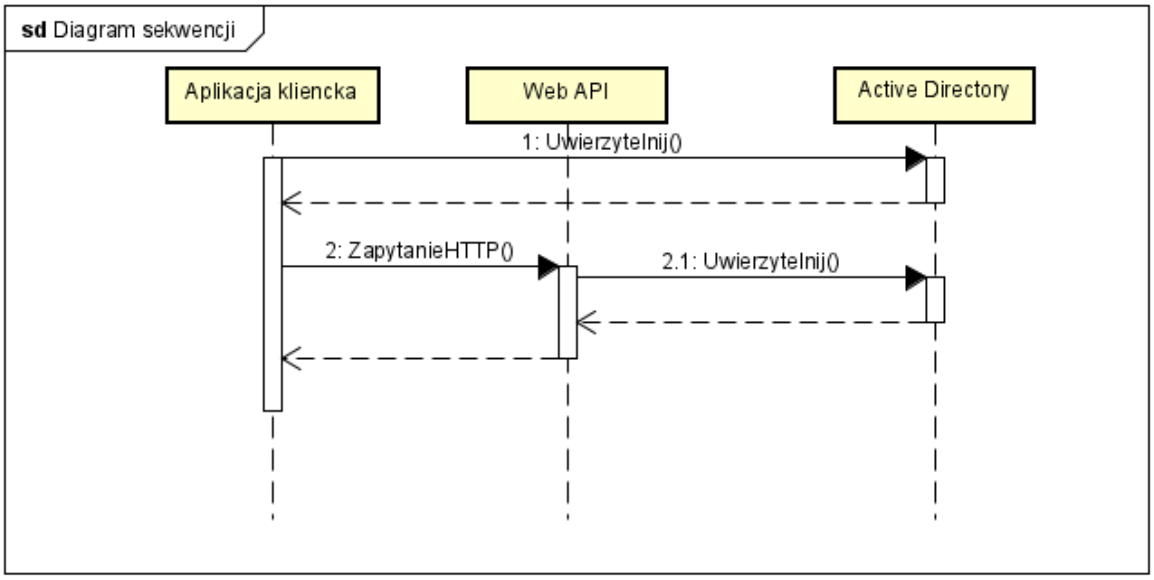
Odczyt danych wykorzystuje komponent Web API który jest ukrytym przed użytkownikiem końcowym, mechanizmem zarządzaniem całym systemem chmurowym. Pośredniczy on między aplikacją kliencką a IoT Hub'em i pozwala na zarządzanie urządzeniami podłączonymi do systemu, dodatkowo w celu autoryzowania dostępu do poszczególnych zasobów systemu wykorzystuje on mechanizmy autoryzacji udostępniane przez komponent Active Directory. Wyjściem na zewnątrz całej infrastruktury chmurowej jest REST API z którego korzysta aplikacja kliencka udostępniająca interfejs użytkownika do pracy z systemem. Przebieg opisanej komunikacji między komponentami jeszcze lepiej obrazują dwa diagramy sekwencji przedstawione na rysunkach 20 i 21.




Źródło: Opracowanie własne przy pomocy narzędzia typu CASE: *Astah UML*

**Rys. 20. Diagram sekwencji komunikacji między komponentami systemu**

Jak widać kluczowym elementem systemu, jest aplikacja webowa w postaci Web API oraz aplikacji klienckiej udostępniającej interfejs użytkownika, to one będą definiować interakcję między systemem gromadzenia danych a użytkownikiem. Pozostałe elementy znajdujące się wewnątrz infrastruktury chmurowej to komponenty udostępniane przez dostawcę usług chmury Azure – firmę Microsoft. Komponenty te są bardzo modularne i gotowe do współpracy ze sobą.




Źródło: Opracowanie własne przy pomocy narzędzia typu CASE: *Astah UML*

**Rys. 21. Diagram sekwencji komunikacji między komponentami - proces uwierzytelniania**

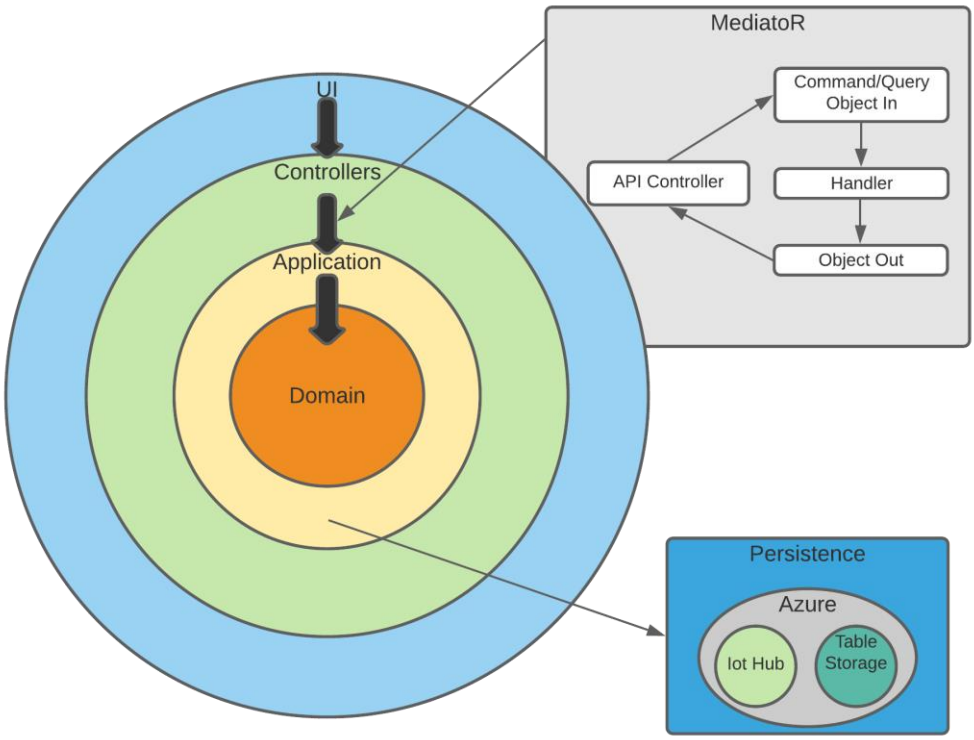
### 2.4.2. Aplikacja internetowa

Aby umożliwić użytkownikom korzystanie z serwisów infrastruktury chmurowej oraz przystosować system do obsługi wielu użytkowników potrzebna jest aplikacja internetowa która zapewni takie możliwości. Aby aplikacja była podatna na utrzymanie oraz rozwijanie w przyszłości musi spełniać pewne założenia.

Architektura aplikacji powinna być zgoda z zasadami *Clean Architecture* (architektura zaproponowana przez Roberta C. Martina w książce o takim samym tytule) a także kod powinien być napisany z zachowaniem zasad SOLID[27]. Na rysunku 22 została przedstawiona proponowana architektura apilikacji.

Architektura składa się z szeregu warstw, w samym centrum znajduje się warstwa domenowa, która modeluje obiekty domeny. Kolejna warstwa - aplikacji, gdzie miejsce ma cała logika biznesowa, to w niej powinny odbywać się wszystkie operacje. Warstwą która obsługuje zapytania HTTP jest warstwa kontrolerów.

Dodatkowo w celu zapewnienia luźnego powiązania klas oraz ograniczenia zależności należy skorzystać z wzorca projektowego o nazwie Mediator – pozwoli on uniezależnić warstwę kontrolerów i warstwę aplikacji od siebie.




Źródło: Opracowanie własne z wykorzystaniem narzędzia Lucidchart w wersji darmowej

**Rys. 22. Architektura aplikacji internetowej (klienckiej)**

Wzorzec mediatora umożliwia luźne powiązanie klas i zmniejszenie ich zależności. Mediator jest klasą która zna metody innych klas i nimi rozporządza. Klasy nie muszą nic o sobie widzieć, a polecenia przekazują jedynie mediatorowi, który jest odpowiedzialny za rozsyłanie tych poleceń do określonych obiektów.

Ostatnią warstwą jest warstwa interfejsu użytkownika, która jest aplikacją uruchamianą w przeglądarce internetowej klienta, i współpracującą w resztą warstw za pomocą protokołu HTTP.

Warto jeszcze wspomnieć o jednym komponencie z którego korzysta warstwa aplikacji, jest nim komponent dostępu do danych *Persistence*, zapewniany przez usługi chmurowe Azure – dokładnie komponenty IoT Hub i Table Storage.

Podsumowując wysokopoziomową architekturę aplikacji internetowej, można powiedzieć, że warstwy *Domain, Application, Controllers* oraz *Persistence* należą do tzw. Backend'u aplikacji, natomiast aplikacja uruchamiana przez przeglądarkę klienta jest Frontend'em.

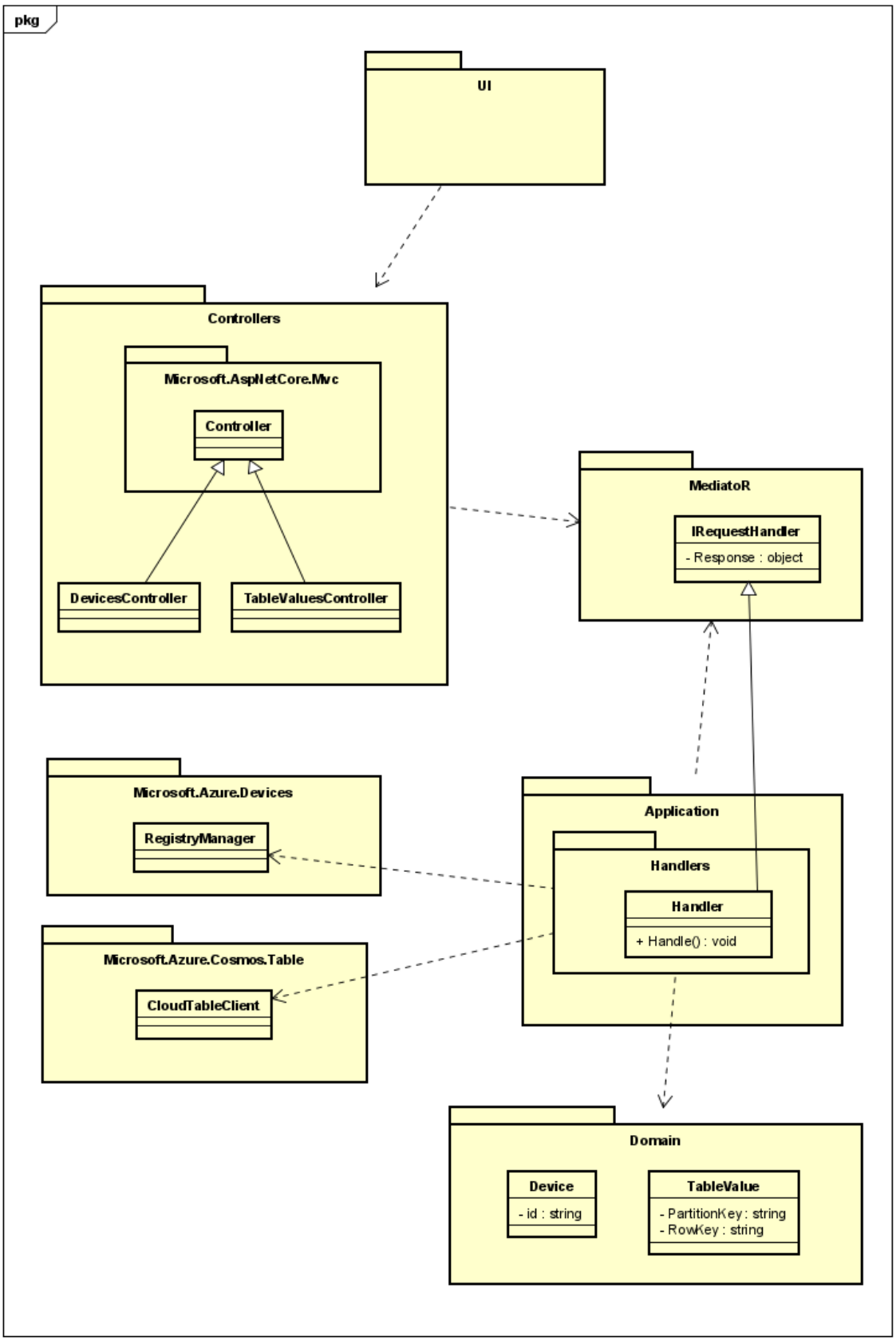
Ze względu na podział warstwowy aplikacji, utworzenie diagramu wszystkich klas byłoby bardzo trudnym zadaniem, dlatego zdecydowałem się utworzyć

uproszczony diagram pakietów i klas obrazujący bardziej szczegółowo projekt aplikacji internetowej – diagram został przedstawiony na rysunku 23. To na co należy zwrócić uwagę to to w jaki sposób aplikacja będzie współpracowała z serwisami chmurowymi, działającymi w chmurze Azure. Za ową współpracę będą odpowiadały dostarczone przez firmę Microsoft, w ramach Azure SDK dla platformy .NET, biblioteki: *Microsoft.Azure.Devices* – biblioteka udostępniająca klasy umożliwiające współpracę z IoT Hub'em oraz *Microsoft.Azure.Cosmos.Table* – biblioteka udostępniająca klasy pozwalające na działanie z przestrzenią przechowywania danych Table Storage.

Samo budowanie aplikacji backendowej w oparciu o platformę .NET 5 wymusza korzystanie z biblioteki klas *Microsoft.AspNetCore.Mvc* udostępniającej klasy do budowy Web API.

Jeżeli chodzi o budowę interfejsu użytkownika, to sposób jego budowy nie jest ściśle powiązany z resztą architektury, a co za tym idzie może zostać zbudowany w dowolnej technologii frontendowej pozwalającej budować aplikacje typu SPA. Ze względu na wcześniejsze doświadczenia z biblioteką React (biblioteka stworzona przez Facebook'a pozwalająca na budowę interfejsu użytkownika z wykorzystaniem języka JavaScript) oraz preferencje silnego typowania i minimalizacje błędów w kodzie, dodatkowo zdecydowałem się wykorzystać do współpracy z biblioteką React język programowania TypeScript będący nadzbiorem JavaScript'u i wprowadzającym do niego silnie typowanie.

Elementem zaprezentowanego diagramu wymagającym dodatkowego komentarza są przerywane strzałki które wskazują na zależności, np. wszystkie elementy działające w pakiecie *Application* mają dostęp do elementów z pakietu *MediatoR*. Oznacza to że może istnieć zależność między klasami z pakietu *Appliaction* a klasami z pakietu *MediatoR*. Ważne jest to, że klasy z pakietu *Application* nie mają bezpośredniego dostępu do elementów działających w obrębie np. pakietu *Controllers*. Podany przykład jest też po części opisem działania wzorca Mediator, dzięki pakietowi *MediatoR* między kontrolerami a logiką biznesowaą została wprowadzona warstwa pośrednicząca co uniezależnia elementy kontrolerów i logiki od siebie.




Źródło: Opracowanie własne przy pomocy narzędzia typu CASE: *Astah UML*

**Rys. 23. Uproszczony diagram pakietów i klas aplikacji internetowej**

### 2.4.3. Specyfikacja REST API

W poprzednim podrozdziale została przedstawiona architektura całej aplikacji internetowej, Web API wraz z aplikacją kliencką. Jak można było zauważyć na diagramie z rysunku 23. to aplikacja frontendowa, udostępniająca interfejs użytkownika, zależy bezpośrednio od kontrolerów – backendu aplikacji webowej. Zgodnie z diagramem komponentów (rysunek 17) komunikacja między interfejsem użytkownika a chmurą przebiega na zasadzie wykorzystania przez aplikację frontendową RESTowego API.

Na etapie projektu należy zdefiniować strukturę budowanego API – udostępniane zasoby oraz punkty końcowe. Punkty końcowe zostały przedstawione w tabeli 17. Modele udostępnianych zasobów, wraz z typem danych poszczególnych pól, zostały przedstawione na listingach 1 i 2.

Reprezentacja obiektu *TableValue* (listing 2.) jest również modelem danych przechowywanych w bazie danych. Chociaż baza typu NoSql nie wymaga ściśle określonego z góry modelu danych, to można przedstawiony model potraktować jako swego rodzaju szablon. Wszystkie operacje bazodanowe w systemie będą przebiegały z wykorzystaniem tego szablonu.

**Tab. 17. Specyfikacja punktów końcowych REST API**

| Punkt końcowy | Typ żądania | Opis |
|---|---|---|
| /api/devices | GET | Uzyskanie wszystkich urządzań zalogowanego użytkownika |
| /api/devices/{device_id} | POST | Utworzenie nowego urządzenia o identyfikatorze `device_id` |
| | PUT | Edycja urządzenia o identyfikatorze `device_id` |
| | DELETE | Usunięcie urządzenia o identyfikatorze `device_id` |
| /api/devices/{device_id}/download | GET | Pobranie pliku konfiguracyjnego dla urządzenia o identyfikatorze `device_id` |
| /api/devices/{device_id}/values | GET | Uzyskanie danych utworzonych, przez urządzenie o identyfikatorze `device_id` |

**Listing 1. Model danych – urządzenia IoT**

```
{
  "devices": [
    {
      "deviceId": "string",
      "deviceType": "string",
      "location": "string",
      "name": "string",
      "temperatureSensor": "boolean",
      "humiditySensor": "boolean",
      "pressureSensor": "boolean",
     "sendFrequency_ms": "integer",
     "connected": "boolean",
     "enabled": "boolean",
     "running": "boolean"
   }]
}
```

**Listing 2. Model danych - wiadomości przechowywanych w bazie danych**

```
{
  "tableValues": [
    {
      "partitionKey": "string",
      "rowKey": "string",
      "timestamp": Date
      "eTag": "string",
      "humidity": "float",
      "pressure": "float",
      "temperature": "float",
      "sentTimestamp": Date
    }]
}
```

# ROZDZIAŁ III.

# IMPLEMENTACJA SYSTEMU

## 3.1. Narzędzia

Narzędzia jakie wykorzystałem do implementacji zostały przedstawione w postaci lisy poniżej, do każdego narzędzia dołączyłem krótki opis zakresu jego wykorzystania w procesie budowy systemu.

- IDE PyCharm

Środowisko deweloperskie tworzone przez firmę JetBrains dla aplikacji tworzonych w języku Python. W projekcie zostało wykorzystane do tworzenia oraz debugowana kodu uruchamianego na urządzeniach Raspberry Pi. Do komunikacji z urządzeniami wykorzystałem protokół SSH który jest bezpośrednio wspierany przez to środowisko, jedynym wymogiem skorzystania z takiego rozwiązania było to, aby urządzenie na którym uruchamiany był kod znajdowało się w tej samej sieci lokalnej co komputer na którym pracowałem.

- Microsoft Azure Portal

Internetowy portal obsługi chmury obliczeniowej Azure udostępniający interfejs użytkownika pozwalający na zarządzenie udostępnianymi usługami.

- IDE Rider

Kolejne środowisko firmy JetBrains, z tym że przeznaczone do tworzenia aplikacji na platformy .NET w języku C#.

- Visual Studio Code

Edytor tekstu stworzony przez firmę Microsoft, dostępny całkowicie za darmo, udostępniający ogromne ilości rozszerzeń pozwalających na dostosowanie środowiska do własnych potrzeb.

- Postman

Narzędzie pozwalające na testowanie punktów końcowych API. Udostępnia interfejs umożliwiający kreowanie zapytań HTTP i podgląd odpowiedzi.

- Sourcetree

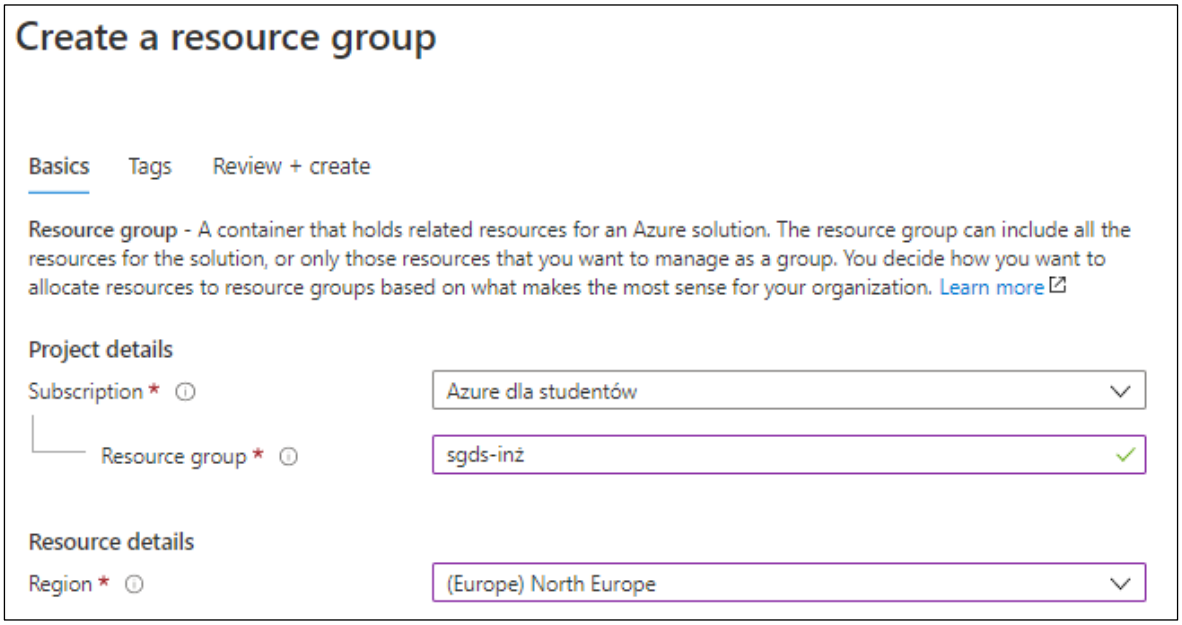
Narzędzie udostępniające interfejs użytkownika pozwalający na prostą obsługę systemu kontroli wersji – git.

## 3.2. Implementacja wybranych komponentów systemu

W celu prezentacji implementacji postanowiłem przedstawić proces tworzenia kluczowych elementów systemu.

### 3.2.1. Przygotowanie przestrzeni chmurowej

W celu skorzystania z serwisów udostępnianych przez chmurę Azure w pierwszej kolejności należało utworzyć grupę zasobów (*resource group)* – jest to swego rodzaju przestrzeń gromadząca serwisy w jeden zespół.




**Rys. 24. Tworzenie grupy zasobów w Azure Portal**

Proces tworzenia grupy zasobów sprowadza się do wypełnienia formularza, ustawienia subskrypcji – czyli planu w ramach jakiego korzystam z platformy Azure, podania nazwy grupy, oraz lokalizacji zasobów.

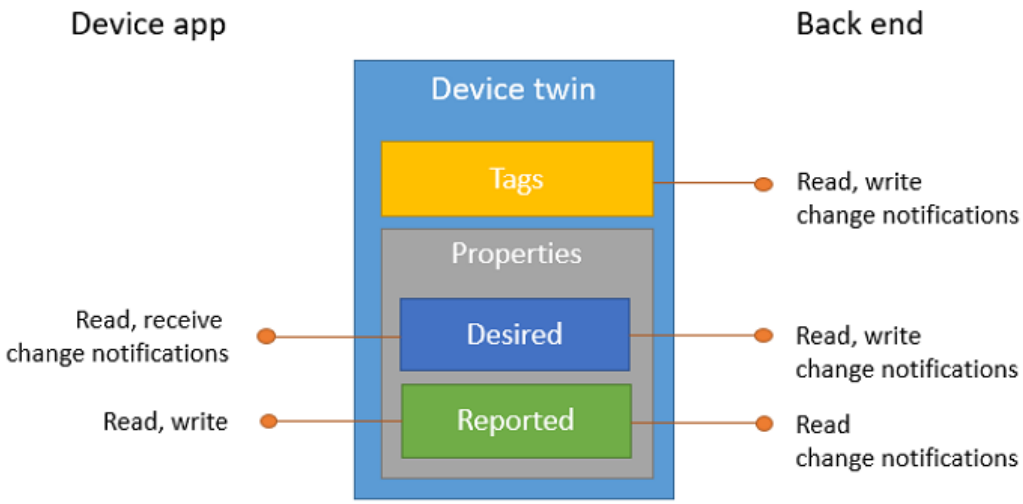
### 3.2.2. Utworzenie serwisu IoT Hub

Serwis IoT Hub należało utworzyć w ramach wcześniej stworzonej przestrzeni. Ponownie utworzenie sprowadzało się do wypełnienia formularza. Interfejs udostępniany przez Azure Portal pozwala na zarządzanie urządzeniami

podłączonymi do Hub'a. Nie jest to jednak metoda odpowiednia dla użytkowników systemu i jedynie administrator powinien mieć dostęp do serwisów działających w chmurze.

Kluczowym dla systemu elementem jest przechowywanie konfiguracji podłączonych urządzeń. Do przechowywania odwzorowania urządzeń po stronie chmury wykorzystałem *Device twins.*

*Device twin* to powiązane z urządzeniem informacje zapisane w formacie JSON. Informacje zawierają szereg właściwości dostępnych zarówno dla samego urządzenia jak i dla aplikacji bekendowej współpracującej z IoT Hub'em[28].




Źródło: [28]

**Rys. 25. Wizualizacja obiektu *Device twin***

Przedstawiona na rysunku 25. wizualizacja obiektu *Device twin* obrazuje widoczność poszczególnych elementów tego obiektu, zarówno dla aplikacji uruchamianej na urządzeniu jak i aplikacji backendowej. W samym obiekcie można wyróżnić 3 zasadnicze rodzaje właściwości:

- Tagi (ang. *Tags)*

Właściwości widziane tylko przez aplikacje backendową, tagi w projekcie zostały wykorzystane do przypisywania urządzeń do użytkowników jak i definiowania własnej nazwy dla urządzenia oraz określania jego lokalizacji. Do tematu tagów wrócę jeszcze na etapie opisywania implementacji Web API, ponieważ to właśnie Web API korzysta z tagów w procesie dodawania nowych urządzeń do systemu[28].

- Właściwości pożądane (ang. *Desired Properties)*

Właściwości jakich aplikacja backendowa żąda od urządzenia. Urządzenie może jedynie odczytać taką właściwość i dostosować swoją konfigurację do określonej właściwości. Zostały wykorzystane do sterowania stanem pracy poszczególnych sensorów urządzenia oraz włączania i wyłączana gromadzenia danych z urządzenia[28].

- Właściwości zaraportowane (ang. *Reported Properties)*

Właściwości możliwe do edycji wyłącznie przez urządzenie, aplikacja backendowa może je jedynie odczytywać dostosować do nich swoją reakcję. Te właściwości nie zostały wykorzystane w implementacji[28].

### 3.2.3. Stream Analytics oraz Table Storage

Utworzenie serwisu Stream Analytics jest procesem analogicznym do procesu tworzenia serwisu IoT Hub. Przed przejściem do konfigurowania procesu przetwarzania danych musiałem utworzyć przestrzeń na dane przychodzące z urządzeń. Tworzenie przestrzeni na dane Table Storage przebiega trochę inaczej, ponieważ serwis ten jest składową tzw. Storage account, który w pierwszej kolejności należało utworzyć. Następnie w przestrzeni Tabel utworzyłem tabelę o nazwie *iotinsights.*

Konfiguracja pracy serwisu Stream Analytics polegała na utworzeniu procedury w języku SQL wykonywanej na strumieniu danych przychodzący. Przed utworzeniem procedury należało zdefiniować aliasy dla danych wejściowych, w tym przypadku IoT Hub'a (alias: *iothub*) oraz dla wyjścia dla danych – tabeli (alias: *sgdstelemetrytable*).

**Listing 3. Procedura przetwarzania danych przez serwis Stream Analytics**

```
1 :  SELECT
2 :      IoTHub.IotHub.MessageId AS MessageId,
3 :      IotHub.IotHub.ConnectionDeviceId AS DeviceId,
4 :      temperature AS Temperature,
5 :      humidity AS Humidity,
6 :      pressure AS Pressure,
7 :      IotHub.IotHub.EnqueuedTime AS SentTimestamp
8 :  INTO
9 :      sgdstelemetrytable
10:  FROM
11:      iothub
```

Z obiektu reprezentującego jedną wiadomość z danymi przychodzącego do serwisu wyciągane są kolejno *MessageId* – identyfikator wiadomości, *DeviceId* – identyfikator urządzenia, z którego nadeszła wiadomość, T*emperature* – dane o zbadanej przez urządzenie temperaturze, *Humidity* – dane o wilgotności powietrza oraz *Pressure* – dane o ciśnieniu atmosferycznym. Dodatkowo aby można było rozpatrywać dane jako szereg czasowy pod polem *SentTimestamp* zapisywany jest czas zakolejkowania wiadomości w IoTHub'ie. Nowoutworzony obiekt zapisywany jest w Tabeli *iotinsights.*

Każda wiadomość w tabeli jest zapisywana jako kolejny wiersz. Na identyfikator wiersza składają się klucz wiersza – *RowKey* oraz klucz partycji *PartitionKey*. Klucz wiersza musi być unikatowy i jest tak naprawdę kluczem głównym każdego zapisanego w bazie obiektu, dlatego zdecydowałem się na wykorzystanie unikalnego identyfikatora wiadomości. Klucz partycji to klucz pomagający na organizację tabeli w partycje – organizacja tabeli w partycje pomaga w szybszym przeszukiwaniu tabel. Najlepszym kandydatem na klucz partycji był identyfikator urządzenia – dzięki temu dane będą pogrupowane względem urządzeń które je przesłały.

### 3.2.4. Program uruchamiany przez urządzenie IoT

Program wykonywany przez urządzenie został napisany w języku Python 3.7. Urządzenia Raspberry Pi zostały wyposażone w płytkę rozszerzeniową *sense HAT* – zostało to szerzej opisane wcześniej, w części projektowej.

W celu zapewnienia asynchroniczności kodu wykorzystałem bibliotekę *asyncio*. W programie można wyróżnić dwa zasadnicze bloki wykonywane właśnie asynchronicznie. Pierwszy to blok zapewniający normalną pracę urządzenia, czyli dokonywanie odczytów pomiarów wartości fizycznych – temperatury i wilgotności powietrza oraz ciśnienia atmosferycznego i przesłanie wiadomości do chmury. Drugi blok to obsługa zdarzenia zmiany odwzorowania urządzenia (*Device twin*) po stronie IoT Hub'a – urządzenie nasłuchuje na taką zmianę i jeżeli otrzyma zdarzenie modyfikuje swoją konfigurację.

**Listing 4. Kluczowe elementy kodu uruchomianego na urządzeniach IoT**

```
1 :  def connection_string():
2 :      filepaths = glob.glob("/boot/*_connectionFile.json")
3 :      f = open(filepaths[0], "r")
4 :      data = json.loads(f.read())
5 :      return data["connectionString"];
6 :
7 :  IOTHUB_CONNECTION_STRING = connection_string()
8 :
9 :  def prepare_message():
10:      values = {}
11:      if device_configuration.temperature_sensor: values["temperature"] =
sense.temperature
12:      if device_configuration.humidity_sensor: values["humidity"] =
sense.humidity
13:      if device_configuration.pressure_sensor: values["pressure"] =
sense.pressure
14:      return json.dumps(values)
15:
16:  async def init_config(device_client: IoTHubDeviceClient):
17:      twin = await device_client.get_twin()
18:      desired = twin['desired']
19:      global device_configuration
20:      device_configuration = Config(desired['send_frequency_ms'], de-
sired['temperature_sensor'], desired['humidity_sensor'], desired['pres-
sure_sensor'], desired['running'])
21:
22:  async def twin_patch_handler(patch):
23:          update_config(patch)
24:
25:  def update_config(patch:dict):
26:      device_configuration.send_frequency_ms = patch['send_frequency_ms']
27:      device_configuration.temperature_sensor = patch['temperature_sen-
sor']
28:      device_configuration.humidity_sensor = patch['humidity_sensor']
29:      device_configuration.pressure_sensor = patch['pressure_sensor']
30:      device_configuration.running = patch['running']
31:
32:  async def send_message(device_client: IoTHubDeviceClient):
33:      while True:
34:          msg_data=prepare_message()
35:          msg = Message(msg_data)
36:          msg.message_id = uuid.uuid4()
37:          msg.content_type="telemetry"
38:          if device_configuration.running:
39:              await device_client.send_message(msg)
40:          await asyncio.sleep(device_configuration.send_fre-
quency_ms/1000)
41:
42:  async def main():
43:      device_client = IoTHubDeviceClient.create_from_connec-
tion_string(IOTHUB_CONNECTION_STRING)
44:      await device_client.connect()
45:      device_client.on_twin_desired_properties_patch_received =
twin_patch_handler
46:      await init_config(device_client)
```

```
47:       await asyncio.gather(send_message(device_client))
48:       await device_client.disconnect()
49:
50:   asyncio.run(main())
```

Pierwszy z opisanych bloków został przedstawiony w listingu 4. linie 32-40 – funkcja `send_message`. Przygotowanie danych do wysłania ma miejsce w funkcji `prepare_message()` (linie 9-14) – wiadomość przygotowywana jest w zależności od stanu konfiguracji urządzenia podchowywanego w obiekcie typu `Config` przedstawionej na listingu 5. Następnie zostaje utworzona wiadomość, zdefiniowane zostają w niej składowe elementy `message_id` oraz `content_type` . Po sprawdzeniu stanu pola `running` w konfiguracji urządzenia, jeżeli jest ono ustawione, wiadomość zostaje wysłana, w przeciwnym razie urządzenie od razu przechodzi do czekania przez zadaną ilość czasu przed ponownym wysyłaniem wiadomości.

**Listing 5. Klasa modelująca konfigurację urządzenia IoT**

```
1:  class Config:
2:      def __init__(self, freq, temp, humi, press, run):
3:          self.send_frequency_ms = freq
4:          self.temperature_sensor = temp
5:          self.humidity_sensor = humi
6:          self.pressure_sensor = press
7:          self.running = run
```

Klasa ta modeluje konfigurację urządzenia. Na konfigurację składają się stany sensorów temperatury, wilgotności oraz ciśnienia, długość interwału czasowego między przesyłaniem kolejnych wiadomości `send_frequency_ms` oraz pole `running` mówiące o tym czy urządzenie ma wysyłać wiadomości czy nie. Wszystkie pola klasy poza polem `send_frequency_ms` które jest zmienną stało liczbową, są binarne.

Blok odpowiedzialny za obsługę zdarzenia zmiany obiektu *Device twin* zaimplementowany został w postaci funkcji `update_config()` - (listing 4. linie 25-30). Funkcja w parametrze przyjmuje obiekt `patch` , który jest opisem zmian jakie zaszły w konfiguracji urządzenia po stronie IoT Hub'a.

Kolejnym elementem jaki należy omówić jest przebieg programu przed rozpoczęciem głównej pętli obsługującej wysyłanie wiadomości oraz reakcje na modyfikacje konfiguracji. Aby urządzenie mogło nawiązać połączenie z IoT Hub'em musi posiadać *connectionString*. Na etapie projektu zdecydowałem, że najlepszym

rozwiązaniem będzie wgranie pliku konfiguracyjnego do pamięci urządzenia. W pliku konfiguracyjnym w formacie JSON przechowywany jest *ConnectionString*.

**Listing 6. Przykładowy obiekt JSON zawierający *ConnectionString***

```
{"connectionString": "HostName=sgds-iot-hub.azure-devices.net;Devi-
ceId=test1234;SharedAccessKey=qw7fvRed5amzb097e+5D4+wfJq2Zx5ZvajJe+QOsZkM="}
```

Odczyt przechowywanego w pamięci urządzenia pliku JSON ma miejsce w funkcji `connection_string()` (listing 4. linie 1-5), funkcja po odczytaniu pliku pozyskuje *ConnectionString* i go zwraca. W linii 7 następuje przypisanie zwróconego klucza do zmiennej globalnej `IOTHUB_CONNECTION_STRING` i od tego momentu interpreter ma do niego dostęp w pamięci programu.

Ostatnim krokiem przed rozpoczęciem głównej pętli programu jest pobranie obiektu *Device twin* z chmury, w funkcji `init_config()`, po pobraniu, następuje deserializacja obiektu, odczytanie właściwości typu *desired* i definicja globalnego obiektu typu `Config`. Na listingu 7. zostały zaznaczone właściwości jakie są istotne dla procesu inicjalizacji pracy urządzenia.

Urządzenia zostały tak skonfigurowane aby zaimplementowany program był uruchamiany w momencie rozruchu w postaci oddzielnego procesu, zaraz po podłączeniu zasilania.

**Listing 7. Przykładowa reprezentacja obiektu *Device twin* w serwisie IoT Hub. Zaznaczone właściwości typu *desired*.**

```
{
  "deviceId": "test_device1",
  "etag": "AAAAAAAAAAM=",
  "deviceEtag": "MTMzNDUzMTYw",
  "status": "enabled",
  "statusUpdateTime": "0001-01-01T00:00:00Z",
  "connectionState": "Disconnected",
  "lastActivityTime": "0001-01-01T00:00:00Z",
  "cloudToDeviceMessageCount": 0,
  "authenticationType": "sas",
  "x509Thumbprint": {
    "primaryThumbprint": null,
    "secondaryThumbprint": null   },
  "modelId": "",
  "version": 4,
  "tags": {
    "owner": "e5c322a7-ffb1-4c7f-b643-ef6a5f27cc4b",
    "location": "Testowa1",
    "device_name": "Test1",
    "device_type": "Raspberry Pi 3B"   },
  "properties": {
    "desired": {
      "send_frequency_ms": 1000,
      "temperature_sensor": false,
      "humidity_sensor": false,
      "pressure_sensor": true,
      "running": false,
      "$metadata": {
      ...
},
    "reported": {
      ...
}
  },
  "capabilities": {
    "iotEdge": false   }
}
```

### 3.2.5. Web API

Web API zostało zaimplementowane zgodnie z zasadami REST. Zasobami udostępnianymi przez API są:

/devices – urządzenia (dodawanie, usuwanie, modyfikacja, odczyt)

/devices/{deviceId}/values – dane zgromadzone przez urządzenie (odczyt)

Dodatkowo aby móc filtrować zgromadzone szeregi czasowe danych, zaimplementowana została obsługa filtrowania od zadanej daty początkowej do daty końcowej.

Postanowiłem zaprezentować jedynie kluczowe elementy kodu Web API, to znaczy, implementację funkcjonalności tworzenia nowego urządzenia oraz generowania pliku JSON z *connectionString'iem*, przeznaczonego do wgrana do pamięci urządzenia IoT.

**Listing 8. Metoda *NewDevice()* klasy *DeviceController***

```
1:  [HttpPost]
2:  public async Task<ActionResult<DeviceResponse>> NewDevice(NewDevice.Com-
mand.DeviceParams commandDeviceParams)
3:  {
4:      return await _mediator.Send(new NewDevice.Command(){Owner = HttpCon-
text.User?.Identity?.Name, deviceParams = commandDeviceParams});
5:  }
```

Na listingu 8. została przedstawiona metoda odpowiedzialna za tworzenie nowego urządzenia, zgodnie z zasadami wzorca projektowego Mediator, w kontrolerze zostaje wydana jedynie komenda do utworzenia nowego urządzenia. Komenda zostaje przesłana do globalnego obiektu `_mediator`, który został wstrzyknięty do obiektu kontrolera dzięki domyślnemu kontenerowi wstrzykiwania zależności dostarczanemu przez framework .NET 5.

Logika biznesowa obsługująca komendę dodawania nowego urządzenia została przedstawiona na listingu 9. W pierwszej kolejności wykorzystując obiekt `_registryManager` i jego metodę `AddDeviceAsync()` (linia 4.) do serwisu IoT Hub zostaje dodane urządzenie o identyfikatorze przesłanym w zapytaniu. (przykładowe zapytanie HTTP typu POST – tworzące nowe urządzenie zostało zaprezentowane w listingu 10.). Następnie, następuje edycja tagów obiektu *Device twin* odpowiadającemu nowoutworzonemu urządzeniu. Zostaje zdefiniowany właściciel urządzenia (aktualnie zalogowany użytkownik), lokalizacja urządzenia, jego nazwa oraz typ. W kolejnych liniach (13-17) ustawiane są właściwości typu *desired*. Na koniec całej operacji obiekt *twin* zostaje zapisany w serwisie IoTHub.

**Listing 9. Funkcja tworząca nowe urządzenie w przestrzeni IoT Hub**

```
1 :  public async Task<DeviceResponse> Handle(Command request, Cancella-
tionToken cancellationToken)
2 :  {
3 :
4 :  await _registryManager.AddDeviceAsync(new Device(request.device-
Params.DeviceId), cancellationToken);
5 :
6 :  var twin = await _registryManager.GetTwinAsync(request.deviceParams.De-
viceId, cancellationToken);
7 :
8 :  twin.Tags["owner"] = request.Owner;
9 :  twin.Tags["location"] = request.deviceParams.Location;
10:  twin.Tags["device_name"] = request.deviceParams.Name;
11:  twin.Tags["device_type"] = request.deviceParams.DeviceType;
12:
13:  twin.Properties.Desired["send_frequency_ms"] = request.device-
Params.SendFrequency_ms;
14:  twin.Properties.Desired["temperature_sensor"] = request.device-
Params.TemperatureSensor;
15:  twin.Properties.Desired["humidity_sensor"] = request.deviceParams.Hu-
miditySensor;
16:  twin.Properties.Desired["pressure_sensor"] = request.deviceParams.Pres-
sureSensor;
17:  twin.Properties.Desired["running"] = request.deviceParams.Running;
18:
19:  var twinResponse = await _registryManager.UpdateTwinAsync(twin.De-
viceId, twin, twin.ETag, cancellationToken);
20:  ...
21:  }
```

**Listing 10. Przykładowe zapytanie HTTP typu POST**

```
POST https://sgds-ddawidzak.azurewebsites.net/api/devices
Authorization: Bearer <bearer-token>
Host: sgds-ddawidzak.azurewebsites.net
Content-Type: application/json;charset=UTF-8
Content-Length: 27

Body
{
    "id":"iot01",
    "deviceType":"Raspberry",
    "location":"WAT",
    "name":"telemetria-sztab",
    "temperatureSensor": true,
    "humiditySensor": true,
    "pressureSensor": true,
    "sendFrequency_ms": 500,
    "running": true
}
```

Generowanie pliku JSON z kluczem *ConnectionString* zostało zaimplementowane również w wykorzystaniem wzorca Mediator. Generowany klucz składa się kolejno z nazwy hosta, którą jest adres do utworzonego serwisu IoT Hub. Ponieważ każde urządzenie łączące się z chmurą korzysta z tego samego serwisu, nazwa hosta została zapisana w pliku konfiguracyjnym projektu, który opiszę jeszcze bardziej szczegółowo przy okazji opisywania współpracy API z Azure Active Directory. Kolejnym elementem klucza jest unikalny identyfikator urządzenia, a ostatnią składową jest klucz symetryczny generowany przez IoT Hub podczas tworzenia urządzenia. Kod odpowiedzialny za zbudowanie i zwrócenie *ConnectionString'a* jest przedstawiony na listingu 11.

**Listing 11. Funkcja tworząca plik zawierający *ConnectionString***

```
1:  public async Task<byte[]> Handle(Query request, CancellationToken can-
cellationToken)
2:  {
3:    var device = await _registryManager.GetDeviceAsync(request.Id, cancel-
lationToken);
4:    var deviceConnectionString = device.Authentication.SymmetricKey.Prima-
ryKey;
5:    var iotHubHostName = _config.GetSection("IotHubConfiguration")["Host-
Name"];
6:    var fileContent = $"{{\"connectionString\": \"HostName={iotHubHost-
Name};DeviceId={request.Id};SharedAccessKey={deviceConnectionString}\"}}";
7:    return Encoding.UTF8.GetBytes(fileContent);
8:  }
```

W pierwszej kolejności w zmiennych `device, deviceConnectionString` oraz `iotHubHostName` zostają zapisane wszystkie składowe elementy potrzebne do utworzenia klucza. W zmiennej `fileContent` następuje złożenie wcześniej pozyskanych elementów w postać jednego ciągu znaków o określonym formacie. Funkcja zwraca wynik swojego działania w postaci tablicy bajtów, aby możliwe było przesłanie pliku w odpowiedzi na zapytanie HTTP.

### 3.2.6. Aplikacja kliencka (React Single Page Application)

Wykorzystanie biblioteki React pozwoliło na budowę aplikacji klienckiej w oparciu o model komponentowy. Do zarządzania stanem aplikacji wykorzystałem bibliotekę MobX pozwalającą na wykorzystanie funkcjonalnego programowania reaktywnego. Dzięki tej bibliotece takie rzeczy jak przeładowanie widoku po zmianie zestawu przechowywanych przez aplikację danych czy pozyskanie tokena dostępu do zasobów przed jego wygaśnięciem dzieją się automatycznie.

Model komponentowy pozwolił na uzyskanie bardzo dużej modularności rozwiązania. Dodatkową wykorzystaną biblioteką był zbiór komponentów graficznych Semantic UI, który pozwala na szybkie zbudowanie przejrzystych i przyjemnych dla oka widoków. Do współpracy z Azure Active Directory wykorzystałem bibliotekę udostępnianą przez firmę Microsoft – MSAL (The Microsoft Authentication Library), wspomagającą budowę funkcjonalności uwierzytelniania oraz autoryzacji w oparciu o protokół OAuth 2.0. Biblioteką jaką wykorzystałem do prezentacji zgromadzonych szeregów czasowych w postaci graficznej jest biblioteka *Rechart* udostępniająca komponenty pozwalające na budowę spersonalizowanych wykresów.

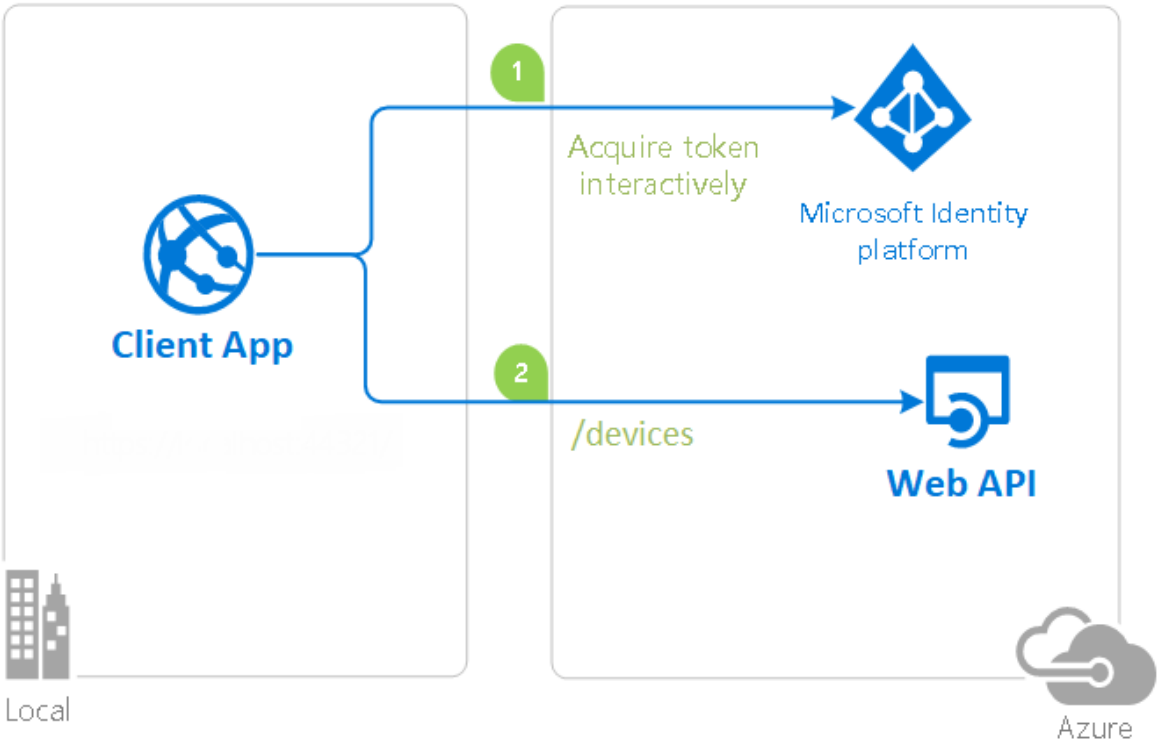
### 3.2.7. Azure Active Directory

Proces tworzenia serwisu Azure Active Directory (AAD) jest analogiczny do tworzenia wszystkich innych serwisów w chmurze Azure i polega na przejściu wszystkich kroków przez które przeprowadza Azure Portal.

Warto zwrócić uwagę na to, że aby umożliwić współpracę Web API oraz aplikacji SPA z AAD należało odpowiednio skonfigurować zarówno serwis chmurowy, jak i obie aplikacje.

Po stronie Azure Portal udostępniony jest interfejs do tworzenia użytkowników, zakresów dostępowych, ról a nawet do edycji interfejsu graficznego ekranu logowania lub rejestracji. Dodatkowo AAD udostępnia możliwość uwierzytelniania w oparciu o konto na innym portalu np. Facebooka czy Google.

Przed prezentacją konfiguracji Web API oraz aplikacji przeglądarkowej do współpracy z AAD, chciałbym jeszcze przedstawić jak uwierzytelnianie i autoryzacja działają w budowanym systemie.




Źródło: https://docs.microsoft.com/en-us/samples/azure-samples/active-directory-aspnetcore-webapp-openidconnect-v2/how-to-secure-a-web-api-built-with-aspnet-core-using-the-azure-ad-b2c/ z modyfikacjami

**Rys. 26. Przebieg uwierzytelniania w systemie**

Aby użytkownik uzyskał dostęp do zasobu z Web API w pierwszej kolejności musi przejść proces uwierzytelniania (np. poprzez zalogowanie się). Na rysunku 26. strzałka nr 1 wskazuje na to że aplikacja kliencka pozyskuje token od AAD. Użytkownik tym sposobem posiada już potwierdzenie dla Web API że jest tym za kogo się podaje. Token otrzymany w odpowiedzi na poprawne logowanie to JWT (JSON Web Token), zawiera on w sobie szereg zaszyfrowanych informacji takich jak: zakres dostępu zalogowanego użytkownika, jego nazwę i np. czas wygaśnięcia tego tokenu. Strzałka nr 2 pokazuje że aplikacja kliencka przesyła zapytanie do Web API, razem z zapytaniem w nagłówku *Authorization* zostaje przesłany token. Web API przeprowadza autoryzacje dostępu, jeżeli użytkownik jaki dokonał zapytania posiada dostęp do zasobu to taki zasób otrzymuje w przeciwnym razie żądanie zostaje odrzucone.

Konfiguracja korzystania z AAD przez Web API sprowadzała się w pierwszej kolejności do utworzenia kolejno polisy logowania i rejestracji oraz polisy zmiany hasła. Polisy tworzy się wykorzystując Azure Portal. Utworzyłem polisy:

`B2C_1_signupsignin1` oraz `B2C_1_passwordreset1`. Następnie należało utworzyć obiekt aplikacji Web API w AAD, dzięki temu obiekt otrzymał swoje Id. Fragment pliku konfiguracyjnego w którym należało zawrzeć zarówno Id obiektu jak i nazwy polis zaprezentowałem na listingu 12.

**Listing 12. Fragment pliku konfiguracyjnego Web API odpowiedzialny za poprawne korzystanie z Azure Active Directory**

```
{
 "AzureAdB2C": {
   "Instance": "https://sgdsddawidziak.b2clogin.com",
   "ClientId": "22888b3a-c663-4932-b8ef-5c1b1e39daa4",
   "Domain": "sgdsddawidziak.onmicrosoft.com",
   "SignUpSignInPolicyId": "B2C_1_signupsignin1",
   "ResetPasswordPolicyId": "B2C_1_passwordreset1"
 },

   ...
}
```

Aplikacja frontendowa wymagała nieco więcej konfiguracji, kod opisujący tą konfiguracje umieściłem w listingu 13.

Dodatkową konfiguracją na jaką należy zwrócić uwagę to linie 14-16 gdzie zdefiniowane zostaje miejsce przechowywania tokena, biblioteka MSAL udostępnia możliwość cache'owania tokena w pamięci lokalnej przeglądarki lub pamięci sesji.

Kolejnym elementem wartym podkreślenia jest zdefiniowany zakres dostępu, do którego o dostęp będzie prosiła aplikacja kliencka przy każdorazowej próbie pozyskania tokenu – jak widać aplikacja prosi o dostęp do zasobów z zakresu `https://sgdsddawidziak.onmicrosoft.com/api/application.access`.
W Web API taki zakres został zdefiniowany w polityce autoryzacji dostępu, pod nazwą *AccessScope*. Jako przykład wykorzystania, na listingu 14. umieściłem metodę z Web API zwracającą wszystkie urządzenia należące do zalogowanego użytkownika, która dzięki atrybutowi `[Authorize(Policy = "AccessScope")]` wymaga właśnie takiego zakresu dostępu od użytkownika wysyłającego zapytanie.

**Listing 13. Kod konfigurujący korzystanie z AAD przez aplikację kliencką**

```
1 :  const tenant = "sgdsddawidziak.onmicrosoft.com";
2 :  const signInPolicy = "B2C_1_signupsignin1";
3 :  const applicationID = "859b4a35-fe39-4304-9460-5102bd70e5a2";
4 :  const tenantSubdomain = tenant.split(".")[0];
5 :  const instance = `https://${tenantSubdomain}.b2clogin.com/tfp/`;
6 :  const signInAuthority = `${instance}${tenant}/${signInPolicy}`;
7 :
8 :  const msalConfig: Configuration = {
9 :    auth: {
10:      clientId: applicationID,
11:      authority: signInAuthority,
12:      knownAuthorities: [signInAuthority],
13:    },
14:    cache: {
15:      cacheLocation: "localStorage",
16:    },
17:  };
18:
19:  export const loginRequest: PopupRequest = {
20:    scopes: ["https://sgdsddawidziak.onmicrosoft.com/api/application.ac-
cess"],
```

**Listing 14. Jedna z metod Web API wymagająca posiadania zakresu dostępu *AccessScope***

```
1:  [HttpGet]
2:  [Authorize(Policy = "AccessScope")]
3:  public async Task<ActionResult<List<DeviceResponse>>> GetAllDevices()
4:  {
5:    var owner = User?.Identity?.Name;
6:    return await _mediator.Send(new GetAll.Query(){Owner = owner});
7:  }
```

### 3.2.8. Azure App Services i publikacja aplikacji

Serwis Azure App Service służy do hostowania aplikacji internetowych. Ponieważ zdecydowałem się na udostępnienie mojej aplikacji w Internecie należało dokonać publikacji aplikacji właśnie do tego serwisu. Publikacja przebiegła z wykorzystaniem narzędzi udostępnianych przez zintegrowane środowisko deweloperskie.

W celu upublicznienia zarówno API jak i aplikacji SPA w ramach jednej instancji Azure App Service zdecydowałem się, na to aby aplikacja Web API pod domyślnym punktem końcowym serwowała statyczne pliki aplikacji klienckiej.

Takim sposobem aplikacja została udostępniona pod adresem: https://sgds-ddawidzak.azurewebsites.net/.

# ROZDZIAŁ IV.

# TESTY SYSTEMU W RAMACH WYBANEGO CASE STUDY

Po zakończeniu pracy nad implementacją, przystąpiłem do przetestowania działania zbudowanego systemu. Do testów zdecydowałem się postawić na miejscu użytkownika końcowego chcącego podłączyć do platformy dwa urządzenia IoT gromadzące dane o temperaturze, wilgotności powietrza oraz ciśnieniu atmosferycznym.

## 4.1. Przypadek testowy

Nowy użytkownik systemu chce zgromadzić dane meteorologiczne z dwóch miejsc – z pomieszczenia w którym przebywa większą część dnia oraz z zewnątrz. Wykorzysta do tego dwa urządzenia Raspberry Pi z płytką rozszerzeniową Sense HAT, które umieści – pierwsze na swoim biurku przy którym pracuje, a drugie na parapecie za oknem. Dzięki gromadzeniu takich danych chciałby dowiedzieć się jaka różnica w temperaturze, ciśnieniu oraz wilgotności powietrza występuje między tymi dwoma lokalizacjami.

## 4.2. Scenariusz przeprowadzenia testu

Scenariusz testowy, będący swego rodzaju instrukcją przeprowadzenia testu został przedstawiony w tabeli 18. Każdy wiersz tabeli jest następującym po sobie krokiem, w odniesieniu do tych kroków zostanie opisany cały proces wykonania scenariusza. Przedstawiona konfiguracja kroków testowych pozwoli na przetestowanie systemu pod kątem każdego przypadku użycia opisanego w części projektowej niniejszej pracy.

**Tab. 18. Opis scenariusz testowego w postaci kroków**

| L.p. | Nazwa kroku |
|---|---|
| 1. | Zakładanie nowego konta użytkownika |
| 2. | Dodanie nowych urządzeń |
| 3. | Pobranie plików konfiguracyjnych dla nowych urządzeń |
| 4. | Wgranie plików konfiguracyjnych do pamięci urządzeń |
| 5. | Rozpoczęcie gromadzenia danych |
| 6. | Wylogowanie z systemu |
| 7. | Zalogowanie do systemu |
| 8. | Modyfikacja konfiguracji jednego z urządzeń |
| 6. | Podgląd zgromadzonych danych |
| 7. | Podgląd analizy danych |
| 8. | Usunięcie urządzeń |

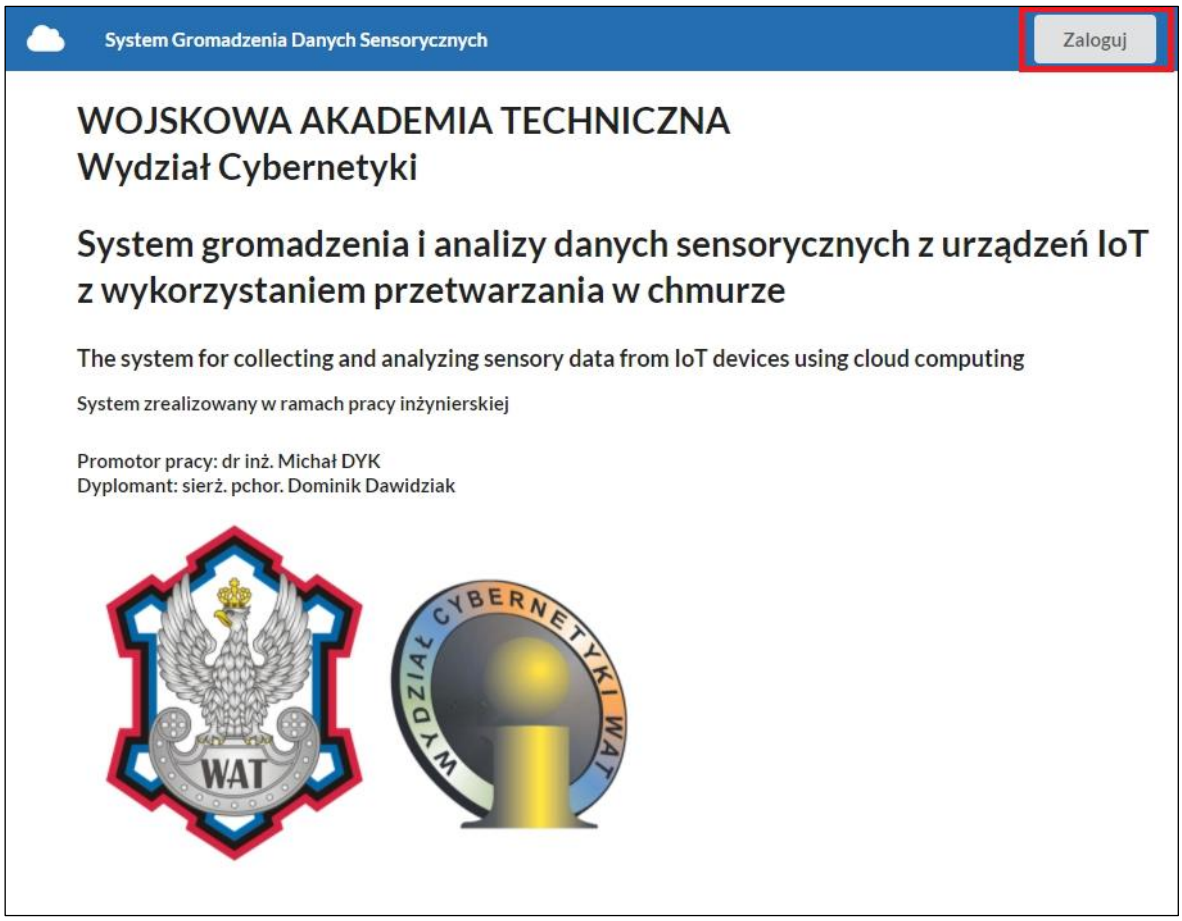
## 4.3. Testy systemu

### 4.3.1. Zakładanie nowego konta

Aby móc korzystać z Systemu Gromadzenia Danych Sensorycznych (SGDS) należy posiadać konto użytkownika. W celu założenia nowego konta użytkownik musi wejść za pomocą przeglądarki internetowej na stronę pod adresem: https://sgds-ddawidzak.azurewebsites.net/.

Oczom użytkownika ukazuje się strona główna systemu, w prawym górnym roku znajduje się przycisk *Zaloguj*, który należy wcisnąć aby móc uzyskać dostęp do systemu. Lokalizacja przycisku zaloguj została zaznaczona na rysunku numer 27.

Na rysunku 29. został przestawiony cały proces zakładania nowego konta użytkownika. Cyfry w kółkach opisują kolejne kroki jakie użytkownik wykonuje, warto zwrócić uwagę na to, że adres e-mail nowego użytkownika należy zweryfikować poprzez wprowadzenie kodu przesłanego automatycznie na podany w procesie rejestracji adres.



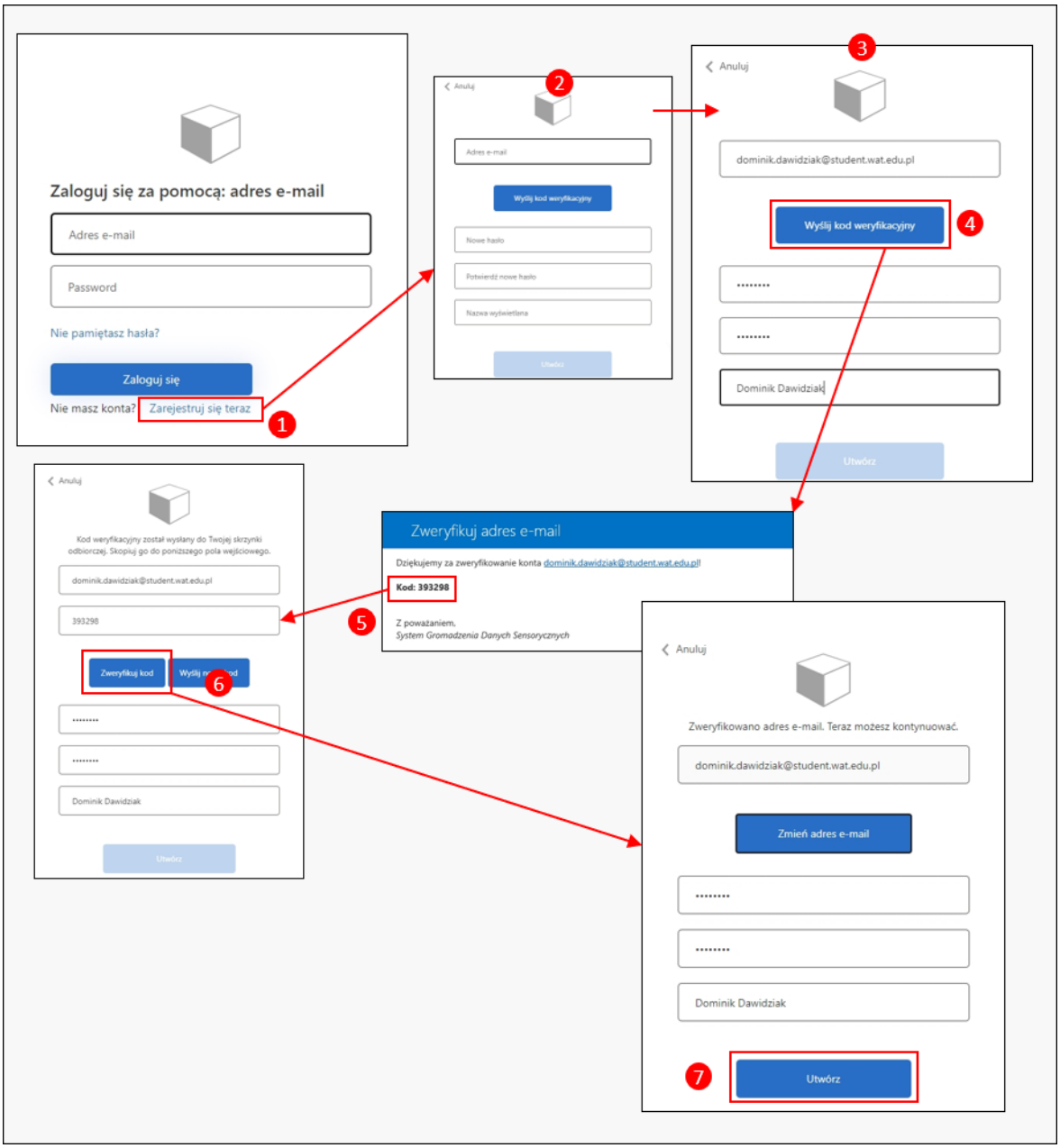

Źródło: Opracowanie własne

**Rys. 27. Strona główna Systemu Gromadzenia Danych Sensorycznych**

Po dokonaniu rejestracji użytkownik zostaje automatycznie zalogowany, a po lewej stronie okna głównego pojawia się boczny pasek udostępniający możliwość korzystania z szeregu funkcjonalności systemu, widok został pokazany na rysunku numer 28.


Źródło: Opracowanie własne

**Rys. 28. Widok zalogowanego użytkownika**

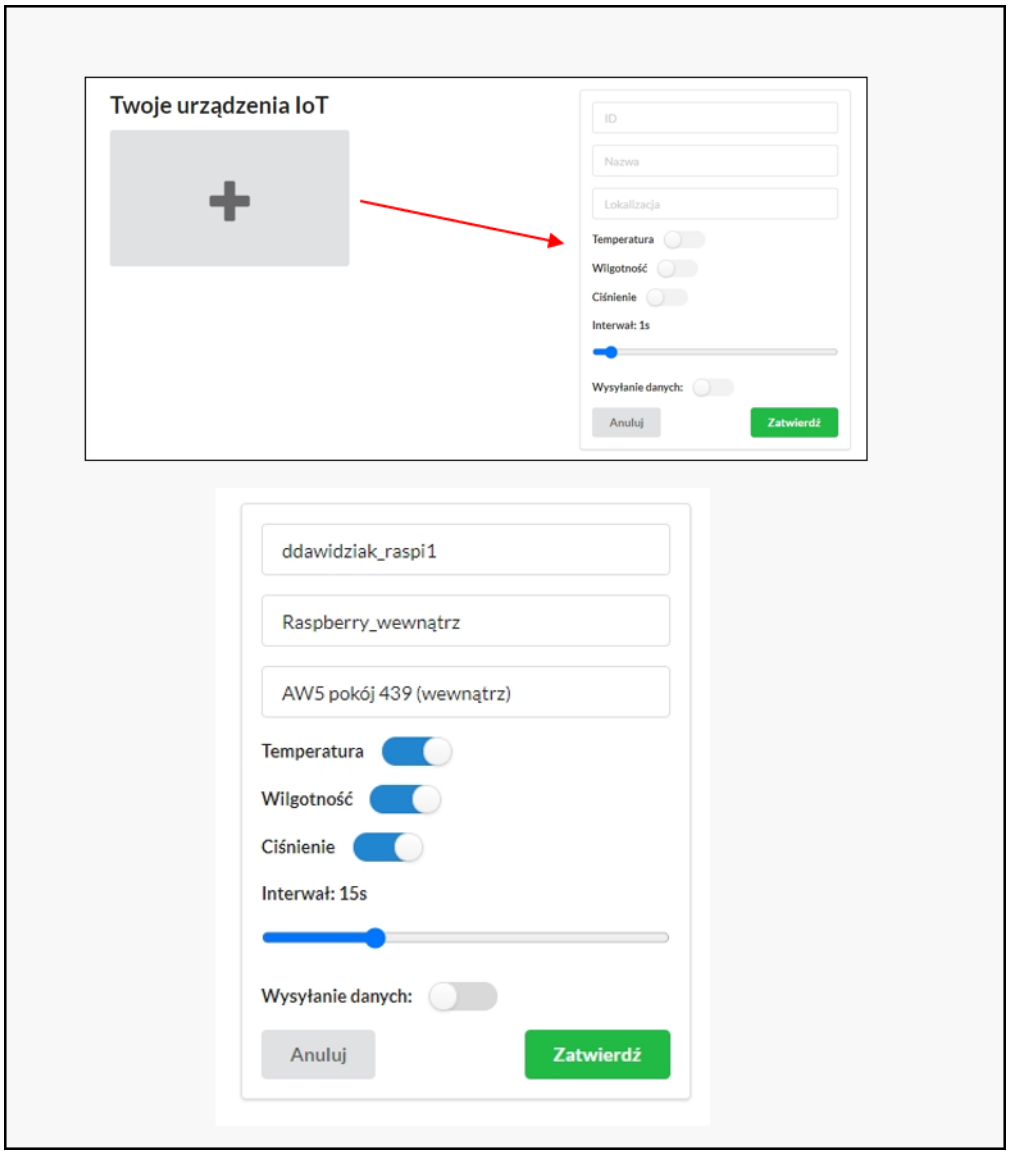



Źródło: Opracowanie własne

**Rys. 29. Proces zakładania konta w Systemie Gromadzenia Danych Sensorycznych**

### 4.3.2. Dodanie nowych urządzeń

Nowy użytkownik nie posiada jeszcze żadnych swoich urządzeń podłączonych do systemu. W celu dodania nowego urządzenia, należy przejść do zakładki *Urządzenia IoT* na bocznym pasku (rysunek 28). Następnym krokiem jest kliknięcie dużego przycisku oznaczonego znakiem „+”. Po kliknięciu przycisku, po prawej stronie ekranu pojawia się formularz dodawania nowego urządzenia który należy wypełnić. Rysunek 30 pokazuje jak wygląda proces dodawania urządzenia.

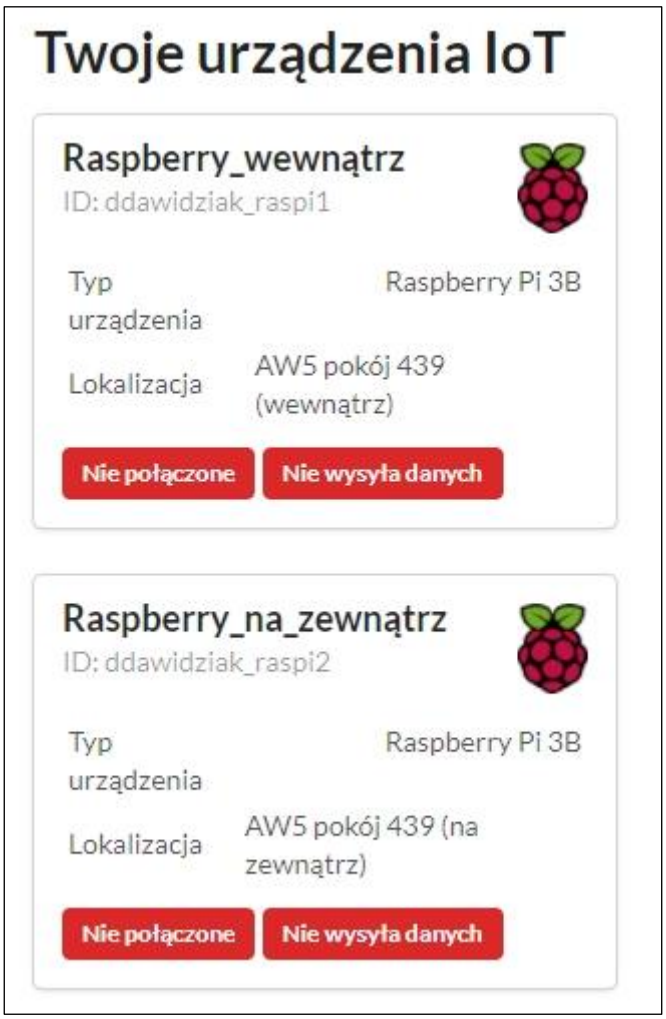



Źródło: Opracowanie własne

**Rys. 30. Dodawanie nowego urządzenia IoT do systemu**

W formularzu należy wprowadzić identyfikator nowego urządzenia (dowolny ciąg znaków), nazwę urządzenia, lokalizację. Następnie ustawić sekcję konfiguracyjną urządzenia. W sekcji konfiguracyjnej można włączać i wyłączać wykonywanie pomiarów poszczególnych wielkości fizycznych oraz ustawić interwał czasu jaki ma następować między przesyłaniem przez urządzenie kolejnych wiadomości. Kluczowym elementem jest przełącznik *Wysyłanie danych,* który jest bezpośrednim włącznikiem gromadzenia danych z wybranego urządzenia, oczywiście warunkiem tego aby dane były gromadzone jest wcześniejsze nawiązane połączenie urządzenia z IoT Hub'em.

W ramach scenariusza testowego użytkownik przeprowadza procedurę dodawania nowego urządzenia dwukrotnie. Rysunek 31. Pokazuję widok urządzeń użytkownika po ukończeniu tego kroku scenariusza.




Źródło: Opracowanie własne

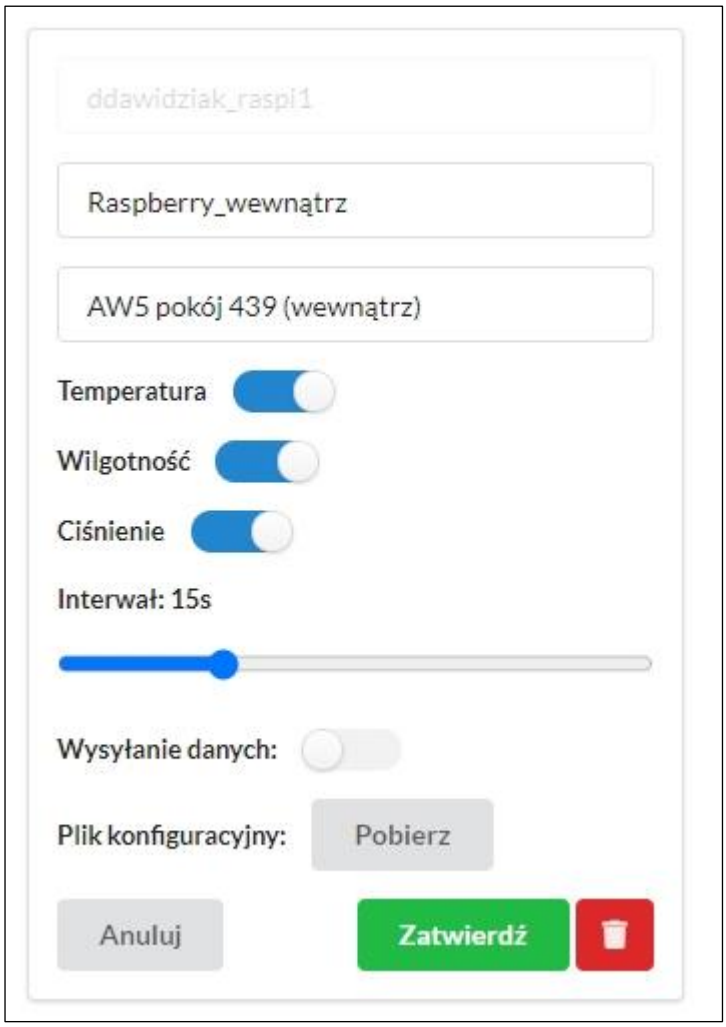
**Rys. 31. Widok urządzeń użytkownika po zakończeniu kroku scenariusza – Dodanie nowych urządzeń**

Jak widać na powyższym rysunku urządzenia zostały nazwane w taki sposób aby nie było problemów z rozróżnieniem ich lokalizacji.

Poza widokiem elementów jakie były konfigurowane na etapie dodawania urządzeń, na karcie każdego urządzenia, można jeszcze zauważyć tagi: *Nie podłączone, Nie wysyła danych* w kolorze czerwonym – które są odzwierciedleniem obecnego stanu urządzeń.

### 4.3.3. Pobranie plików konfiguracyjnych dla nowych urządzeń

Pobieranie pliku konfiguracyjnego dla dodanego urządzenia jest możliwe w każdym momencie pracy systemu. Po kliknięciu w kartę urządzenia, analogicznie do formularza dodawania, po prawej stronie ukazuje się bardzo podobny formularz lecz wzbogacony właśnie o przycisk pobierania pliku konfiguracyjnego, oraz przycisk usuwania urządzenia. Widok formularza edycji został przedstawiony na rysunku 32.




Źródło: Opracowanie własne

**Rys. 32. Widok formularza edycji urządzenia**

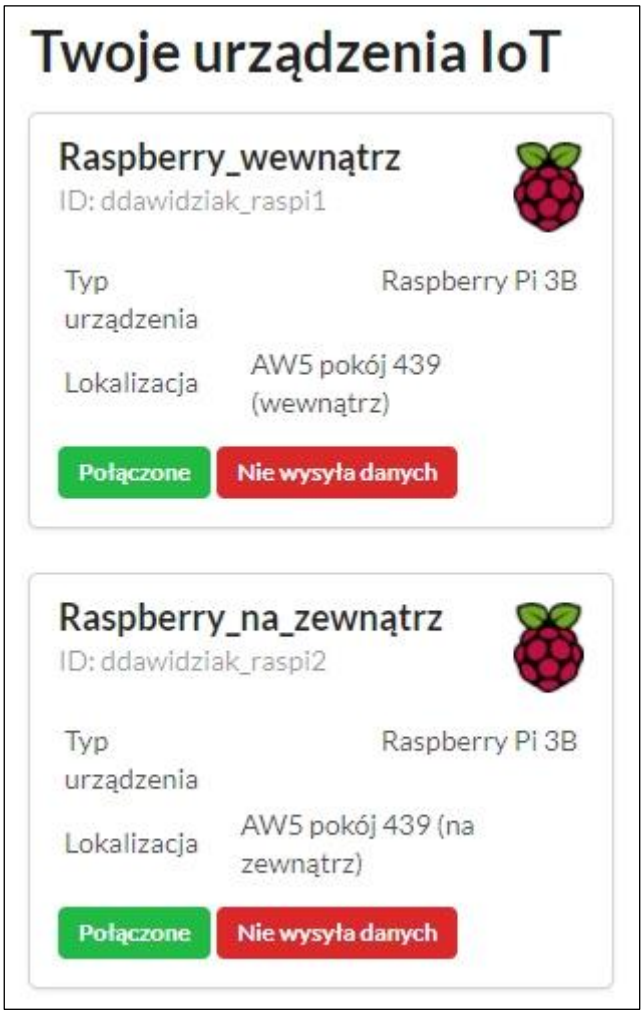
Po kliknięciu przycisku *Pobierz* przeglądarka automatycznie wyświetla okno dialogowe, w którym użytkownik wybiera lokalizację zapis pobieranego pliku. Nazwy pobranych plików to: `ddawidziak_raspi1_connectionFile.json` oraz `ddawidziak_raspi2_connectionFile.json`.

### 4.3.4. Wgranie plików konfiguracyjnych do pamięci urządzeń

Wgranie plików konfiguracyjnych do pamięci urządzeń jest procesem bardzo prostym. Użytkownik musi jedynie skopiować plik konfiguracyjny na kartę pamięci urządzenia Raspberry Pi.

### 4.3.5. Rozpoczęcie gromadzenia danych

Jak zostało wspomniane wcześniej aby móc rozpocząć gromadzenie danych przez urządzenia, między nimi a IoT Hub'em musi zostać nawiązane połączenie. Połączenie zostaje nawiązywane automatycznie po włączeniu zasilania urządzenia. Kilka sekund po włączeniu zasilania urządzeń, użytkownik może zaobserwować zmianę stanu w zakładce *Urządzenia IoT*. Widok urządzeń które nawiązały połączenie został pokazany na rysunku 33.




Źródło: Opracowanie własne

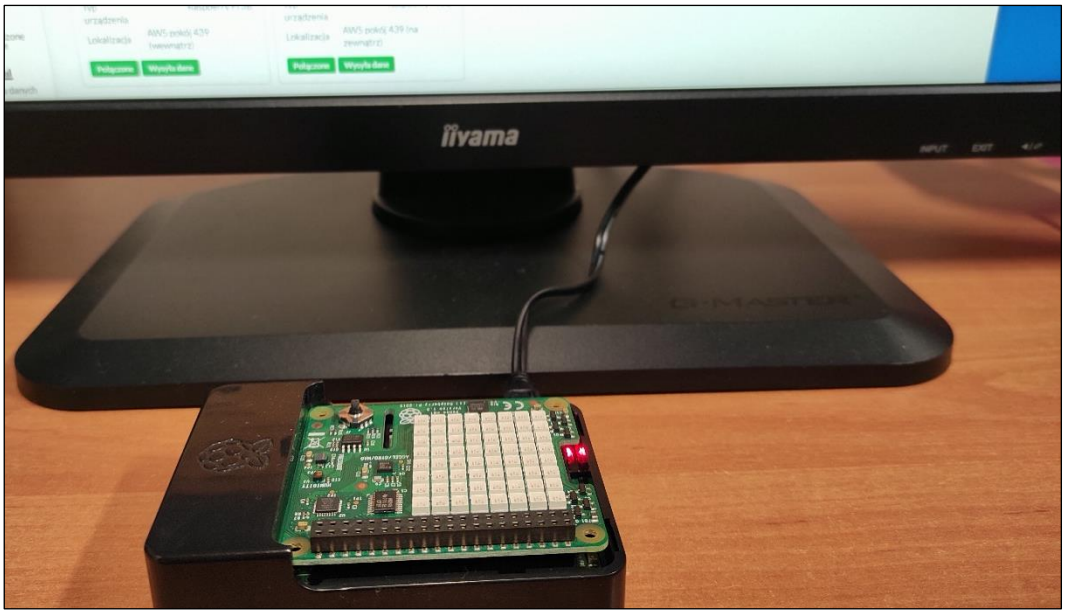
**Rys. 33. Widok urządzeń w systemie po nawiązaniu połączenia**

Ponieważ w formularzu edycji (rysunek 32) opcja *Gromadzenie danych* nie została wcześniej zaznaczona, widać że na tym etapie urządzenia nawiązały połączenie z chmura, lecz nie rozpoczęły jeszcze procesu przesyłania danych. Po zmianie stanu przełącznika *Gromadzenie danych* dla obydwu urządzeń, następuje zmiana stanu, która odzwierciedla rysunek 32. Od tego momentu dane z urządzeń są gromadzone w przestrzeni chmurowej.


Źródło: Opracowanie własne

**Rys. 34. Widok urządzeń wysyłających dane**

W ramach lepszej wizualizacji procesu testowania postanowiłem dodatkowo na rysunkach nr 35 i 36 pokazać jak wyglądała fizyczna konfiguracja i lokalizacja urządzeń podczas przeprowadzania testów.




Źródło: Opracowanie własne

**Rys. 35. Urządzenie Raspberry_wewnątrz (id: ddawidziak_raspi1) zlokalizowane na biurku**

Źródło: Opracowanie własne

**Rys. 36. Urządzenie Raspberry_na_zewnątrz (id: ddawidziak_raspi2) zlokalizowane na parapecie za oknem**

### 4.3.6. Wylogowanie z systemu

Ponieważ dane będą gromadzone przez całą noc nie ma takiej potrzeby aby użytkownik był przez ten czas zalogowany do systemu. W celu wylogowania należy wcisnąć przycisk w prawym górnym rogu (widoczny na rysunku nr 28). Po wylogowaniu użytkownik zostaje przekierowany do strony głównej systemu.

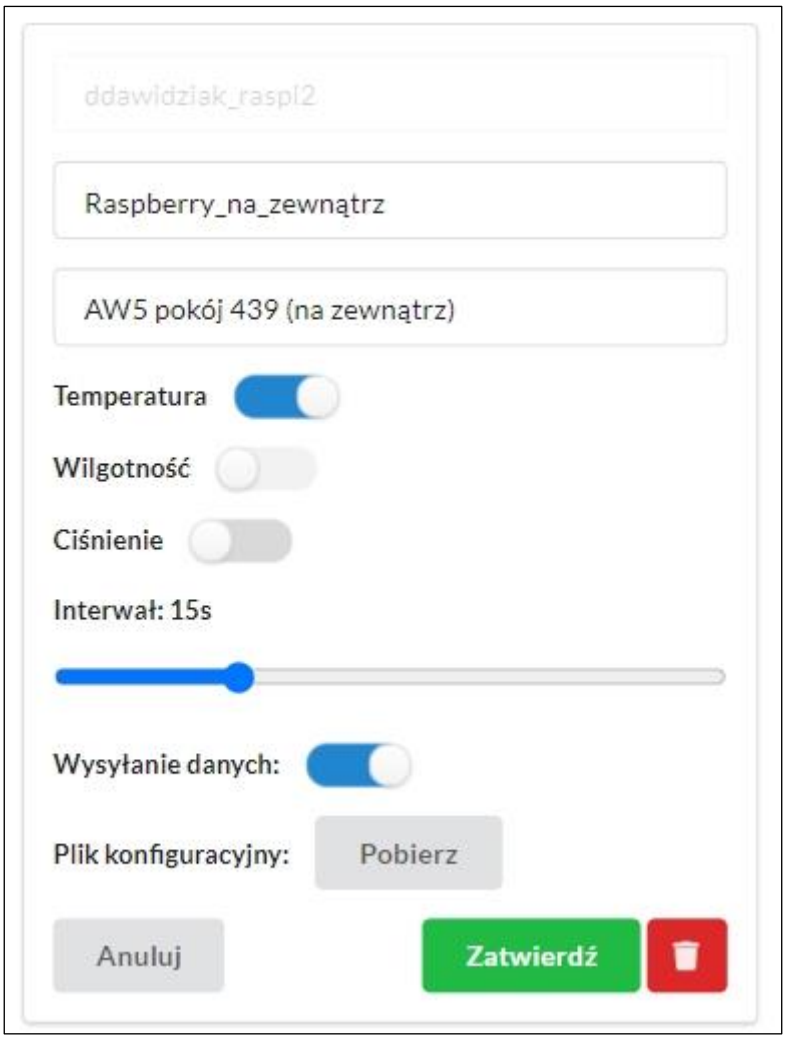
### 4.3.7. Logowanie do systemu

Po upływie czasu jaki został przeznaczony na gromadzenie danych w ramach prowadzonego testu należało się zalogować. Logowanie rozpoczyna się identycznie jak proces rejestracji, przedstawiony w punkcie 4.3.1. W celu uwierzytelnienia należy podać adres e-mail podany podczas zakładania konta oraz hasło. Weryfikacja adresu e-mail nie jest przeprowadzana podczas tej procedury.

### 4.3.8. Modyfikacja konfiguracji jednego z urządzeń

Na potrzeby testów wszystkich funkcjonalności, została przeprowadzona modyfikacja konfiguracji jednego z podłączonych urządzeń, a dokładnie Raspberry_na_zewnątrz. Odczyty z sensorów wilgotności oraz ciśnienia zostały w tym urządzeniu wyłączone. Rysunek 37. pokazuje widok formularza edycji urządzenia po zmianie konfiguracji.

Efekty zmiany konfiguracji zostaną przedstawione i omówione w kolejnym punkcie scenariusza testowego.



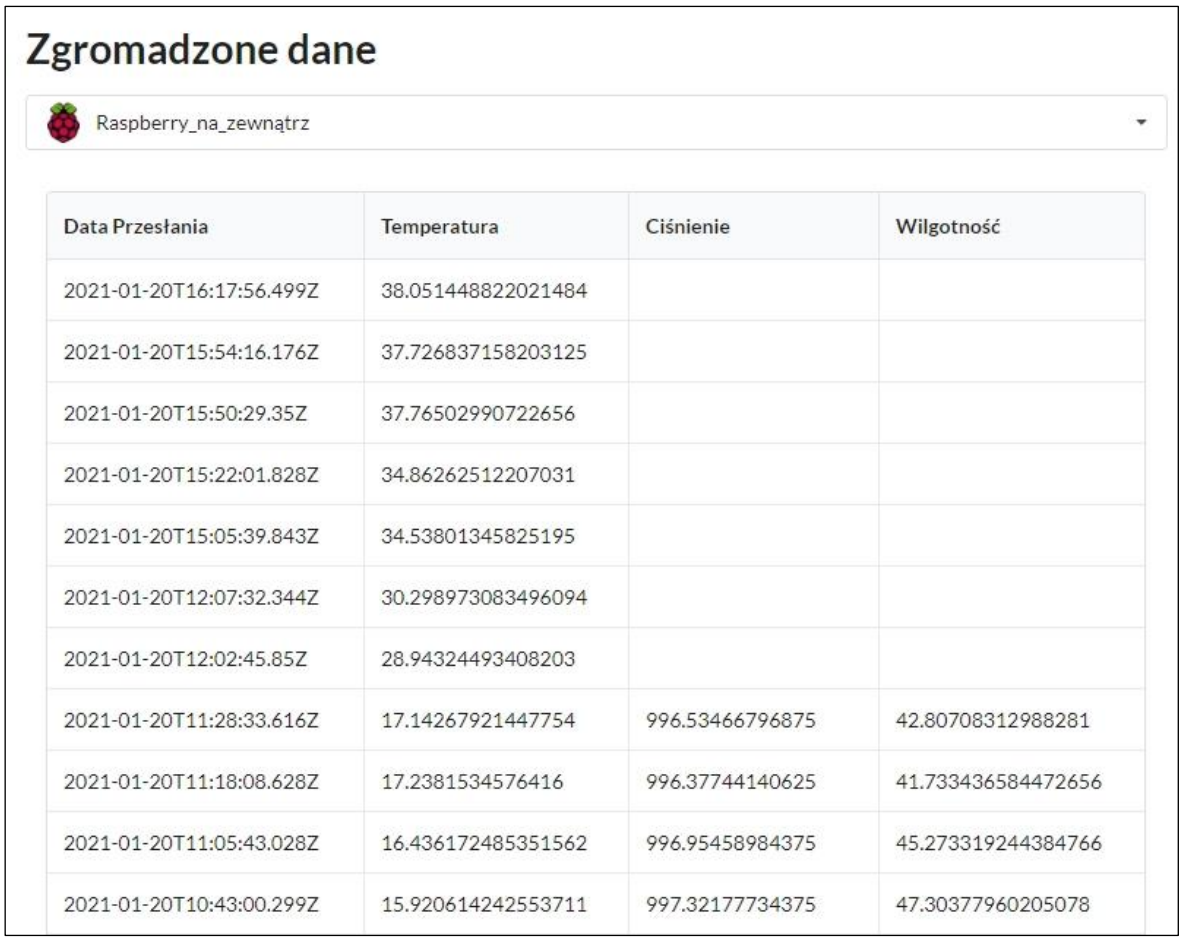

Źródło: Opracowanie własne

**Rys. 37. Modyfikacja konfiguracji urządzenia Raspberry_na_zewnątrz**

### 4.3.9. Podgląd zgromadzonych danych

W zakładce *Zgromadzone dane* użytkownik ma dostęp do podglądu zgromadzonych danych w postaci surowej. Dane wyświetlane są w postaci tabeli. Prezentowane jest 50 ostatnich otrzymanych od urządzenia odczytów. Kolumny tabeli w jakiej dane są prezentowane to:

- Data przesłania - w postaci znacznika czasu w formacie UTC - (*Universal Time Coordinated* – Uniwersalny czas koordynowany),
- Temperatura – dane temperaturowe w stopniach Celsjusza,
- Ciśnienie – dane o ciśnieniu atmosferycznym w hektopaskalach,
- Wilgotność – dane o wilgotności powietrza w %.




| Data Przesłania | Temperatura | Ciśnienie | Wilgotność |
|---|---|---|---|
| 2021-01-20T16:17:56.499Z | 38.051448822021484 | | |
| 2021-01-20T15:54:16.176Z | 37.726837158203125 | | |
| 2021-01-20T15:50:29.35Z | 37.76502990722656 | | |
| 2021-01-20T15:22:01.828Z | 34.86262512207031 | | |
| 2021-01-20T15:05:39.843Z | 34.53801345825195 | | |
| 2021-01-20T12:07:32.344Z | 30.298973083496094 | | |
| 2021-01-20T12:02:45.85Z | 28.94324493408203 | | |
| 2021-01-20T11:28:33.616Z | 17.14267921447754 | 996.53466796875 | 42.80708312988281 |
| 2021-01-20T11:18:08.628Z | 17.2381534576416 | 996.37744140625 | 41.733436584472656 |
| 2021-01-20T11:05:43.028Z | 16.436172485351562 | 996.95458984375 | 45.273319244384766 |
| 2021-01-20T10:43:00.299Z | 15.920614242553711 | 997.32177734375 | 47.30377960205078 |

Źródło: Opracowanie własne

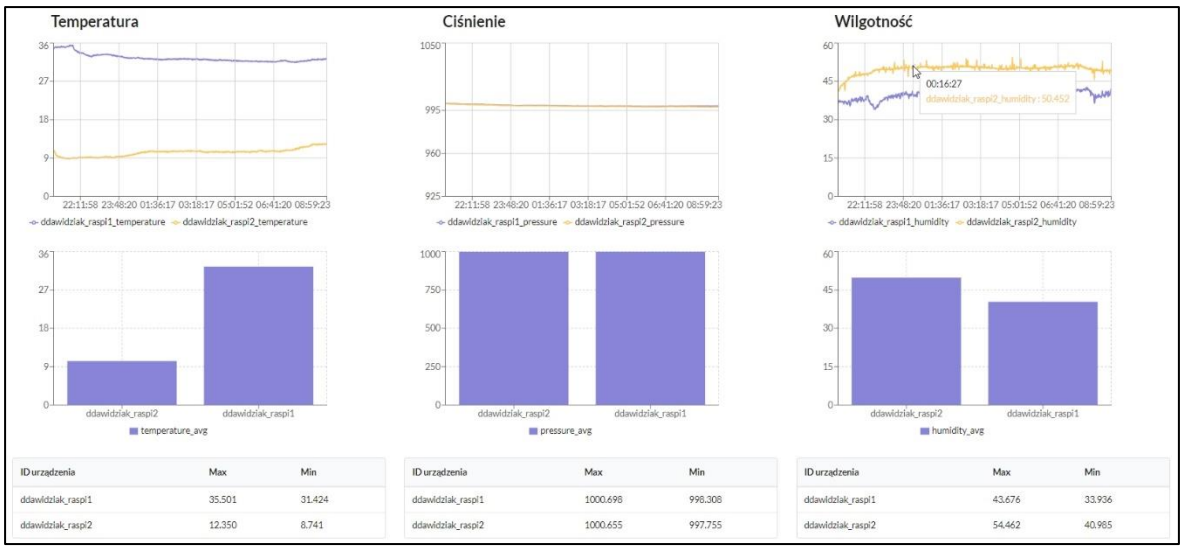
**Rys. 38. Podgląd surowych danych zgromadzonych przez urządzenie Raspberry_na_zewnątrz**

Rysunek 38. jest zrzutem ekranu z zakładki *Zgromadzone dane*, jak widać brakuje danych dla o ciśnieniu i wilgotności dla ostatnich kilku pomiarów, jest to wynik zmiany konfiguracji przeprowadzonej w poprzednim kroku scenariusza testowego. Jest to zjawisko całkowicie pożądane, ponieważ świadczy o poprawności działania funkcji edycji konfiguracji urządzenia IoT w czasie jego pracy.

### 4.3.10. Podgląd analizy danych

Podgląd analizy zgromadzonych danych jest jedną z kluczowych funkcjonalności systemu. Po procesie gromadzenia danych, użytkownik chcąc przeanalizować zgromadzone szeregi czasowe ma do dyspozycji zakładkę *Analiza danych* a w niej możliwość wyboru zestawu danych z jednego lub kilku ze swoich urządzeń. W przypadku wyboru danych z kilku urządzeń są one umieszczane na tych samych wykresach w celu łatwego porównania. Dodatkową opcją na którą warto zwrócić uwagę

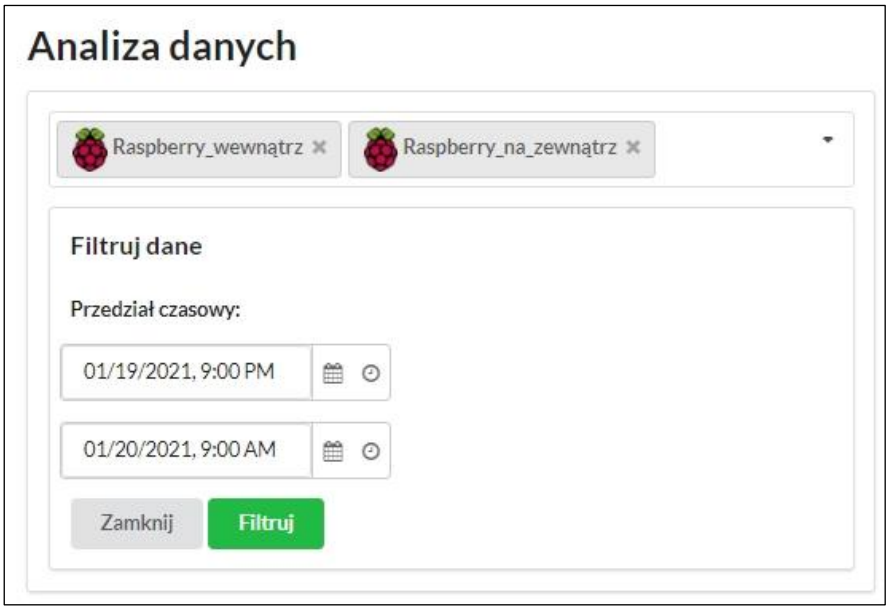
jest możliwość filtrowania danych i wyświetlania analiz dla danych zgromadzonych między ustawioną w filtrach, datą początkową a datą końcową. Widok analiz danych dla urządzeń gromadzących danych podczas testów został pokazany na rysunku 39. Widok filtrów jakim zostały poddane zgromadzone dane umieszczony jest na kolejnym rysunku.




Źródło: Opracowanie własne

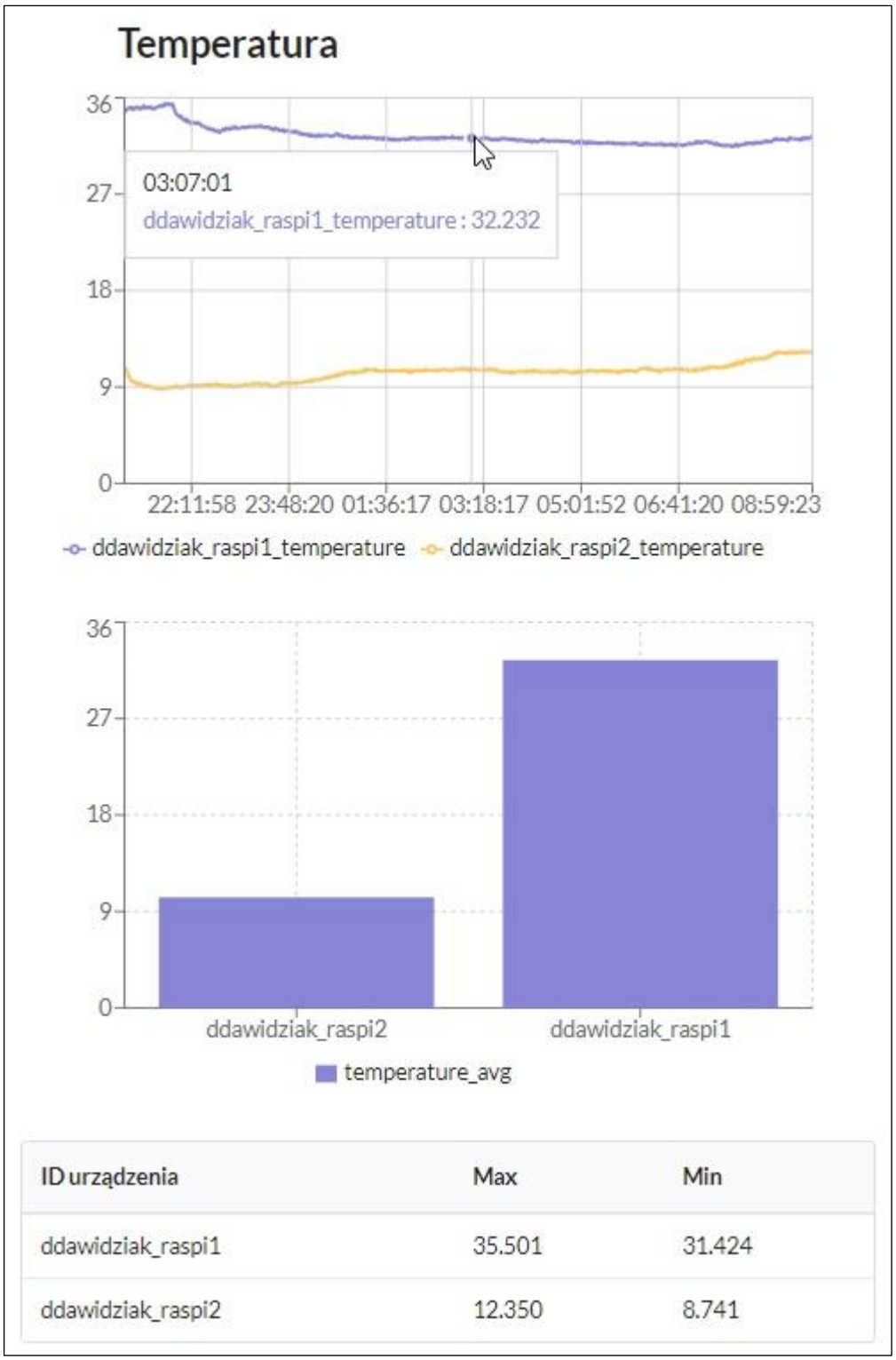
**Rys. 39. Widok analiz zgromadzonych danych**

Ze względu na ograniczone wymiary kartki A4 na rysunku nr 41 umieściłem powiększenie analiz dla danych dotyczących temperatury powietrza, w celu lepszego zobrazowania przykładowej analizy.




Źródło: Opracowanie własne

**Rys. 40. Widok ustawienia filtrów zgromadzonych danych**




| ID urządzenia | Max | Min |
|---|---|---|
| ddawidziak_raspi1 | 35.501 | 31.424 |
| ddawidziak_raspi2 | 12.350 | 8.741 |

Źródło: Opracowanie własne

**Rys. 41. Szczegółowy widok analizy zgromadzonych danych dot. temperatury powietrza**

Dodatkową opcją dostępną dla użytkownika podczas analizowania danych jest możliwość wyświetlenia szczegółów jednego pomiaru. Na powyższym obrazku widać, że po najechaniu kursorem na linie odwzorowującą zbadane wartości temperatury, obok strzałki ukazuje się pole kontekstowe w którym znajdują się szczegóły dotyczące tego jednego pomiaru (godzina i wartość pomiaru).

#### 4.3.11. Usunięcie urządzeń

Ostatnim krokiem scenariusza testowego jest usunięcie urządzeń, w tym celu użytkownik musi jedynie wcisnąć przycisk usuwania w formularzu edycji urządzenia (widoczny w prawym dolnym rogu formularza – rysunek 37). Po kliknięciu przycisku następuje usunięcie wybranego urządzenia – jego karta znika z widoku urządzeń w zakładce *Urządzenia IoT*.

## 4.4. Wnioski

Przeprowadzony scenariusz testowy pozwolił na zweryfikowanie poprawności zaimplementowania wszystkich opisanych w rozdziale projektowym funkcjonalności. System działa poprawnie i spełnia wszystkie postawione wcześniej założenia.

Jedynym mankamentem rzucającym się w oczy, który tak naprawdę nie jest zależny od zbudowanego systemu a od samej płytki rozszerzeniowej Sense HAT, to odczyty temperatury – są one obarczone bardzo wysokim błędem pomiarowym rzędu nawet 10 stopni. Jest to wynik tego że w sąsiedztwie sensora temperatury znajduje się procesor urządzenia Raspberry Pi który w momencie pracy urządzenia emituje ciepło. Właśnie z tego powodu na rysunku 41 można zauważyć, że średnia temperatura w pomieszczeniu wynosiła ok. 32 stopnie a na zewnątrz ok. 10 stopni, co jest wynikiem mocno zawyżonym.

Nie mniej, podłączenie innego rodzaju czujników do Raspberry Pi lub podłączenie nawet innego typu urządzeń nie stanowiłoby najmniejszego problemu i wymagałoby konfiguracji jedynie ze strony urządzenia, podczas gdy system pozostałby niezmienny. Przedstawiona analiza wyników, mimo odchyleń od wartości prawdziwych bardzo dobrze oddaje różnice parametrów wewnątrz i na zewnątrz pomieszczenia.

# ZAKOŃCZENIE

Zrealizowany w ramach pracy System Gromadzenia Danych Sensorycznych, jest propozycją wykorzystania technologii chmurowych dostarczanych przez chmurę Microsoft Azure do gromadzenia i analizy danych z urządzeń IoT. Budowa infrastruktury w oparciu o serwisy z zestawu Azure IoT Suite oraz integracja jej z nowoczesną aplikacją webową opartą o podział między frontend i backend, pozwoliła na uzyskanie produktu wysoce skalowalnego i podatnego na dalsze rozwijanie.

Największym problemem z jakim musiałem się zmierzyć podczas budowania systemu była konfiguracja komunikacji IoT Hub – Urządzenie, ponieważ wymagało to znalezienia możliwości asynchronicznego nasłuchiwania na zmianę konfiguracji urządzenia po stronie chmury. Z pomocom przyszła dokumentacja bibliotek udostępnianych przez firmę Microsoft.

Funkcjonalnościami nad jakimi należałoby się skupić w przyszłości jest dynamiczne mapowanie czujników, tak aby każde urządzenie, bez względu na to w jakie czujniki będzie zaopatrzone, mogło zostać podłączone do systemu. Taka funkcjonalność pozwoliłaby na uniknięcie potrzeby ręcznej modyfikacji konfiguracji urządzenia aby dopasować je do zdefiniowanego przez system interfejsu. Poza tym, można by rozważyć budowę dodatkowej aplikacji mobilnej pozwalającej na sterowanie i monitoring urządzeń. System działa w oparciu o REST API więc budowa aplikacji mobilnej nie byłaby problemem.

Śmiało mogę stwierdzić, że wszystkie założenia pracy zostały zrealizowane i zbudowany system odpowiada na wymagania stawiane mu zarówno we wstępie do tej pracy jak i w jej części projektowej.

# BIBLIOGRAFIA

[1] B. Marr, “How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read.” https://www.forbes.com/sites/bernard-marr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=75f1bed60ba9; 2021.

[2] “On-premises software.” https://en.wikipedia.org/wiki/On-premises_software; 2021

[3] S. Li, L. Da Xu, and S. Zhao, “The internet of things: a survey,” *Inf. Syst. Front.*, vol. 17, no. 2, pp. 243–259, 2015, doi: 10.1007/s10796-014-9492-7.

[4] S. Klein, *IoT Solutions in Microsoft ’ s Azure IoT Suite*.

[5] M2M World of Connected Services, “The Internet of Things http://www.beechamresearch.com/article.aspx?id=4”; 2020

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015, doi: 10.1109/COMST.2015.2444095.

[7] “Internet of Things Architecture | IoT-A Project | FP7 | CORDIS | European Commission.” https://cordis.europa.eu/project/id/257521 (accessed Jun. 09, 2020).

[8] V. Schmidt, “Impact Analysis of the Internet of Things on the Value Chain in Manufacturing Industries,” no. July, 2016.

[9] D. Priyadarshi and A. Behura, “Analysis of Different IoT Protocols for Heterogeneous Devices and Cloud Platform,” *Proc. 2018 IEEE Int. Conf. Commun. Signal Process. ICCSP 2018*, pp. 868–872, 2018, doi: 10.1109/ICCSP.2018.8524531.

[10] M. Stusek, K. Zeman, P. Masek, J. Sedova, and J. Hosek, “IoT Protocols for Low-power Massive IoT: A Communication Perspective,” *Int. Congr. Ultra Mod. Telecommun. Control Syst. Work.*, vol. 2019-Octob, 2019, doi:

10.1109/ICUMT48472.2019.8970868.

[11] C. Prehofer, “Models at REST or modelling RESTful interfaces for the Internet of Things,” *IEEE World Forum Internet Things, WF-IoT 2015 - Proc.*, pp. 251–255, 2015, doi: 10.1109/WF-IoT.2015.7389061.

[12] A. Raza, A. A. Ikram, A. Amin, and A. J. Ikram, “A review of low cost and power efficient development boards for IoT applications,” *FTC 2016 - Proc. Futur. Technol. Conf.*, no. December, pp. 786–790, 2017, doi: 10.1109/FTC.2016.7821693.

[13] Yoppy, R. H. Arjadi, H. Candra, H. D. Prananto, and T. A. W. Wijanarko, “RSSI Comparison of ESP8266 Modules,” *2018 Electr. Power, Electron. Commun. Control. Informatics Semin. EECCIS 2018*, pp. 150–153, 2018, doi: 10.1109/EECCIS.2018.8692892.

[14] Raspberry Pi documentation “https://www.raspberrypi.org/documentation/.” ; 2020.

[15] V. C. Emeakaroha, N. Cafferkey, P. Healy, and J. P. Morrison, “A Cloud-Based IoT Data Gathering and Processing Platform,” *Proc. - 2015 Int. Conf. Futur. Internet Things Cloud, FiCloud 2015 2015 Int. Conf. Open Big Data, OBD 2015*, pp. 50–57, 2015, doi: 10.1109/FiCloud.2015.53.

[16] Leading cloud infrastructure providers “https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.”;2020 .

[17] IBM Cloud Computing “https://www.ibm.com/pl-pl/cloud/learn/cloud-computing-gbl.”; 2020 .

[18] What is IoT Platform “https://www.kaaproject.org/blog/what-is-iot-platform.”; 2020 .

[19] “lora-alliance.” https://lora-alliance.org/about-lorawan/; 2021.

[20] B. Nakhuva and T. Champaneria, “Study of Various Internet of Things Platforms,” *Int. J. Comput. Sci. Eng. Surv.*, vol. 6, no. 6, pp. 61–74, 2015, doi: 10.5121/ijcses.2015.6605.

[21] Azure IoT Suite “https://azure.microsoft.com/en-us/blog/microsoft-azure-iot-suite-connecting-your-things-to-the-cloud/.”; 2020 .

[22] Amazon IoT Core “https://www.amazonaws.cn/en/iot-core/features/.”; 2020 .

[23] Google IoT Core “https://cloud.google.com/iot-core.”; 2020 .

[24] High availability, “https://en.wikipedia.org/wiki/High_availability#Percentage_calculation.”; 2021 .

[25] Azure IoT Hub, “https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub.”; 2020 .

[26] IoT Hub MQTT support “https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support.”; 2020.

[27] R. C. M. (“Uncle Bob”), *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*. Pearson; 1st edition (September 10, 2017).

[28] IoT Hub Device twins “https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-device-twins.”; 2020 .

# Spis rysunków

Rys. 1. Technologie powiązane z IoT[3] ..... 11
Rys. 2. Architektury rozwiązań IoT. (a) Trójwarstwowa (ang. *Three-layer*). (b) Bazująca na Warstwie Pośredniej (ang. *Middle-ware based*). (c) Zorientowana na usługi (ang. *SOA based*). (d) Pięciowarstwowa (ang. *Five-layer*)[6]..... 15
Rys. 3. Model przesyłania wiadomości za pomocą protokołu MQTT - przykład dla danych o temperaturze ..... 17
Rys. 4. Model przesyłania wiadomości za pomocą protokołu AMQP..... 18
Rys. 5. Model przesyłania wiadomości za pomocą protokołu HTTP ..... 19
Rys. 6. Diagram współpracy środowiska CoAP z REST ..... 20
Rys. 7. Przykładowa płytka - Raspberry Pi 3 Model B ..... 22
Rys. 8. Płytka rozwojowa WeMos D1 mini wraz z opisem wyprowadzeń ..... 23
Rys. 9. Udziały w światowym rynku wiodących dostawców usług chmurowych w drugim kwartale 2020 roku..... 24
Rys. 10. Platforma IoT jako warstwa pośrednicząca (ang. middleware) ..... 26
Rys. 11. Stos technologiczny chmurowych platform IoT..... 28
Rys. 12. Usługi dostępne w ramach platformy Azure IoT Suite ..... 29
Rys. 13. Usługi dostępne w ramach platformy AWS IoT ..... 30
Rys. 14. Usługi dostępne w ramach platformy Google Cloud IoT ..... 31
Rys. 15. Diagram przypadków użycia..... 33
Rys. 16. Diagram wysokopoziomowej architektury Systemu gromadzenia i analizy danych sensorycznych z urządzeń IoT w wykorzystaniem przetwarzania w chmurze ..... 45
Rys. 17. Diagram komponentów Systemu gromadzenia i analizy danych sensorycznych z urządzeń IoT w wykorzystaniem przetwarzania w chmurze ..... 49
Rys. 18. Diagram sekwencji komunikacji między komponentami Urządzenie IoT a IoT Hub ..... 50
Rys. 19.Diagram klas programu uruchomianego na Urządzeniu IoT ..... 51
Rys. 20. Diagram sekwencji komunikacji między komponentami systemu ..... 52
Rys. 21. Diagram sekwencji komunikacji między komponentami - proces uwierzytelniania ..... 53

Rys. 22. Architektura aplikacji internetowej (klienckiej) .....54
Rys. 23. Uproszczony diagram pakietów i klas aplikacji internetowej .....56
Rys. 24. Tworzenie grupy zasobów w Azure Portal .....60
Rys. 25. Wizualizacja obiektu *Device twin* .....61
Rys. 26. Przebieg uwierzytelniania w systemie .....72
Rys. 27. Strona główna Systemu Gromadzenia Danych Sensorycznych .....77
Rys. 28. Widok zalogowanego użytkownika .....77
Rys. 29. Proces zakładania konta w Systemie Gromadzenia Danych Sensorycznych .....78
Rys. 30. Dodawanie nowego urządzenia IoT do systemu.....79
Rys. 31. Widok urządzeń użytkownika po zakończeniu kroku scenariusza – Dodanie nowych urządzeń .....80
Rys. 32. Widok formularza edycji urządzenia.....81
Rys. 33. Widok urządzeń w systemie po nawiązaniu połączenia .....82
Rys. 34. Widok urządzeń wysyłających dane .....83
Rys. 35. Urządzenie Raspberry_wewnątrz (id: ddawidziak_raspi1) zlokalizowane na biurku.....84
Rys. 36. Urządzenie Raspberry_na_zewnątrz (id: ddawidziak_raspi2) zlokalizowane na parapecie za oknem.....84
Rys. 37. Modyfikacja konfiguracji urządzenia Raspberry_na_zewnątrz .....86
Rys. 38. Podgląd surowych danych zgromadzonych przez urządzenie Raspberry_na_zewnątrz .....87
Rys. 39. Widok analiz zgromadzonych danych .....88
Rys. 40. Widok ustawienia filtrów zgromadzonych danych.....88
Rys. 41. Szczegółowy widok analizy zgromadzonych danych dot. temperatury powietrza .....89

# Spis tabel

Tab. 1. Standardy komunikacyjne i sensoryczne znajdujące zastosowanie dziedzinie Internetu Rzeczy..... 13
Tab. 2. Opis przypadku użycia UC1 ..... 34
Tab. 3. Opis przypadku użycia UC2 ..... 35
Tab. 4. Opis przypadku użycia UC3 ..... 36
Tab. 5. Opis przypadku użycia UC4 ..... 37
Tab. 6. Opis przypadku użycia UC5 ..... 38
Tab. 7. Opis przypadku użycia UC6 ..... 39
Tab. 8. Opis przypadku użycia UC7 ..... 39
Tab. 9. Opis przypadku użycia UC8 ..... 40
Tab. 10. Opis przypadku użycia UC9 ..... 41
Tab. 11. Opis przypadku użycia UC10 ..... 41
Tab. 12. Opis przypadku użycia UC11 ..... 42
Tab. 13. Opis przypadku użycia UC12 ..... 42
Tab. 14. Opis przypadku użycia UC13 ..... 43
Tab. 15. Opis przypadku użycia UC14 ..... 43
Tab. 16. Specyfikacja wymagań pozafunkcjonalnych..... 44
Tab. 17. Specyfikacja punktów końcowych REST API ..... 57
Tab. 18. Opis scenariusz testowego w postaci kroków..... 76

# Spis listingów

Listing 1. Model danych – urządzenia IoT ..... 58
Listing 2. Model danych - wiadomości przechowywanych w bazie danych ..... 58
Listing 3. Procedura przetwarzania danych przez serwis Stream Analytics ..... 62
Listing 4. Kluczowe elementy kodu uruchomianego na urządzeniach IoT ..... 64
Listing 5. Klasa modelująca konfigurację urządzenia IoT ..... 65
Listing 6. Przykładowy obiekt JSON zawierający *ConnectionString* ..... 66
Listing 7. Przykładowa reprezentacja obiektu *Device twin* w serwisie IoT Hub. Zaznaczone właściwości typu *desired.* ..... 67
Listing 8. Metoda *NewDevice()* klasy *DeviceController* ..... 68
Listing 9. Funkcja tworząca nowe urządzenie w przestrzeni IoT Hub ..... 69
Listing 10. Przykładowe zapytanie HTTP typu POST ..... 69
Listing 11. Funkcja tworząca plik zawierający *ConnectionString* ..... 70
Listing 12. Fragment pliku konfiguracyjnego Web API odpowiedzialny za poprawne korzystanie z Azure Active Directory ..... 73
Listing 13. Kod konfigurujący korzystanie z AAD przez aplikację kliencką ..... 74
Listing 14. Jedna z metod Web API wymagająca posiadania zakresu dostępu *AccessScope* ..... 74