POLITECHNIKA ŁÓDZKA

Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki

Instytut Automatyki

**PRACA DYPLOMOWA MAGISTERSKA**

Projekt i realizacja systemu Internet Rzeczy w oparciu o chmurę obliczeniową

(Design and implementation of Internet of Things System based on cloud computing model)

Marcin Jahn

214331

Opiekun pracy:

dr inż. Jarosław Kacerka

Łódź, Lipiec 2018

Spis treści

[1 Wstęp 3](#_Toc520411494)

[1.1 Cel i zakres pracy 4](#_Toc520411495)

[1.2 Przegląd rozwiązań obecnych na rynku 6](#_Toc520411496)

[1.2.1 IFTTT 6](#_Toc520411497)

[1.2.2 Microsoft Flow 7](#_Toc520411498)

[1.2.3 Ubidots 8](#_Toc520411499)

[2 Usługi chmurowe 9](#_Toc520411500)

[2.1 Dostępne serwisy 9](#_Toc520411501)

[2.2 Dostawcy usług chmurowych 10](#_Toc520411502)

[2.2.1 IBM Watson 11](#_Toc520411503)

[2.2.2 Microsoft Azure 11](#_Toc520411504)

[2.2.3 Amazon AWS 12](#_Toc520411505)

[2.3 PaaS, IaaS oraz SaaS 12](#_Toc520411506)

[2.4 Usługi IoT 13](#_Toc520411507)

[2.4.1 Azure IoT Hub 13](#_Toc520411508)

[2.4.2 AWS IoT Core 14](#_Toc520411509)

[2.5 Wybór dostawcy 14](#_Toc520411510)

[3 Modelowanie oraz łączenie urządzeń 15](#_Toc520411511)

[3.1 Definiowanie urządzeń 15](#_Toc520411512)

[3.2 Dziedziczenie właściwości 17](#_Toc520411513)

[3.3 Łączenie urządzeń 19](#_Toc520411514)

[3.3.1 Kalkulacje 20](#_Toc520411515)

[4 Omówienie projektu 21](#_Toc520411516)

[4.1 Schemat platformy 21](#_Toc520411517)

[4.2 Komponenty umieszczone w chmurze 22](#_Toc520411518)

[4.2.1 IoT Hub 22](#_Toc520411519)

[4.3 Baza urządzeń Azure SQL 25](#_Toc520411520)

[4.3.1 Skalowanie serwisów 25](#_Toc520411521)

[4.3.2 Struktura bazy danych 26](#_Toc520411522)

[5 Bibliografia 27](#_Toc520411523)

# Wstęp

Internet Rzeczy (ang. *IoT – Internet of Things*) to termin, który w ostatnich latach zyskał wielką popularność - słyszał o nim niemal każdy. Nawet osoby niezwiązane profesjonalnie z technologią kojarzą go i są w stanie mniej więcej wyjaśnić na czym polega. Wynika to z faktu, że dziedzina Internetu Reczy cieszy się obecnie ogromnym zainteresowaniem - zarówno ze strony producentów sprzętu elektronicznego, jak i jego nabywców. Powodem tego trendu jest fakt, że idea Internetu Rzeczy okazuje się być przydatna w ogromnej ilości zastosowań. Obecnie implementacje IoT znaleźć można w szerokiej gamie produktów konsumenckich oraz w zastosowaniach profesjonalnych, w przemyśle. WSTAWIĆ INFO O JAKICHŚ BADANIACH. Mimo popularności samego terminu, warto w niniejszej pracy, która traktuje przecież o budowie systemu IoT, przedstawić definicję Internetu Rzeczy. Powinno to stanowić pewnego rodzaju fundament, na podstawie którego wyjaśniane będą założenia i funkcjonalność opisywanego projektu.

Okazuje się, że niełatwo jest przedstawić definicję, która w pełni opisywałaby zagadnienie. Jako dowód tego niech posłuży 86-stronicowy dokument przygotowany przez IEEE (ang. *Institute of Electrical and Electronics Engineers*) [1]. Publikacja ta została stworzona jako próba przedstawienia czym jest IoT, z różnych punktów widzenia. Spośród setek definicji, które zawiera, wybrane zostały dwie, które w moim odczuciu trafnie i wystarczająco opisują temat na potrzeby niniejszego projektu:

The Internet of Things (IoT) consists of things that are connected to the Internet, anytime, anywhere. In its most technical sense, it consists of integrating sensors and devices into everyday objects that are connected to the Internet over fixed and wireless networks. The fact that the Internet is present at the same time everywhere makes mass adoption of this technology more feasible. Given their size and cost, the sensors can easily be integrated into homes, workplaces and public places. In this way, any object can be connected and can ‘manifest itself’ over the Internet. Furthermore, in the IoT, any object can be a data source. This is beginning to transform the way we do business, the running of the public sector and the day-to-day life of millions of people. [2]

The Internet of Things refers to the unique identification and ‘Internetization’ of everyday objects. This allows for human interaction and control of these ‘things’ from anywhere in the world, as well as device-to-device interaction without the need for human involvement. [3]

Ostatecznie więc, na podstawie przedstawionych cytatów, można wyróżnić kilka punktów, które charakteryzują system Internetu Rzeczy:

* obecność „rzeczy” (urządzeń, sensorów, itd.), które mają stałe połączenie z Internetem;
* łatwość obsługi oraz dostępność elementów składających się na system;
* każda końcówka systemu stanowi źródło danych;
* sterowanie urządzeń przez Internet, bez bezpośredniego kontaktu człowieka;
* możliwość komunikacji między urządzeniami.

W związku z tym, że IoT może być definiowane na tak wiele sposobów, nie dziwi różnorodność oferowanych rozwiązań, które z tej idei korzystają. Jednocześnie jednak można zwrócić uwagę na fakt, że sama etykieta ‘IoT’ niewiele mówi nam o tym jakie funkcjonalności oferuje dany produkt. Czy chodzi o wysyłanie danych telemetrycznych? Czy może dane urządzenie potrafi komunikować się z innymi urządzeniami? A może jeszcze coś innego? Internet Rzeczy może reprezentować szeroki wachlarz rzeczywistych możliwości danego urządzenia i sam opisywany termin jest zbyt ogólny by był wystarczający do opisu konkretnego produktu czy usługi. W związku z powyższym, kolejny podrozdział ma na celu wyjaśnienie czym właściwie ma być rezultat niniejszej pracy magisterskiej.

## Cel i zakres pracy

Celem pracy jest zaprojektowanie oraz praktyczna realizacja architektury systemu Internet of Things (IoT) bazującego na wykorzystaniu chmury obliczeniowej. System ma umożliwić tworzenie wirtualnych połączeń pomiędzy urządzeniami sterującymi oraz wykonawczymi. Każde z urządzeń zostanie stworzone w oparciu o mikrokontroler wyposażony w łączność wi-fi, w celu komunikacji obustronnej z częścią systemu znajdującą się w chmurze. System powinien pozwolić na dostęp wielu użytkowników, gdzie każdy ma swoją własną pulę urządzeń oraz zestaw reguł z nimi związanych. Konfiguracja urządzeń powinna odbywać się poprzez dedykowaną dla rozwiązania aplikację, dostępną z poziomu smartfona/tabletu bądź komputera.

Powyższy akapit to opis pracy przedstawiony przy zgłaszaniu tematu dla Komisji. Wymaga on jednak pewnego rozszerzenia.

Budowany system (dalej często nazywany „platformą”) ma bazować na wykorzystaniu chmury obliczeniowej. Oznacza to, że główna funkcjonalność i logika projektu ma być umieszczona „w chmurze”, czyli na zewnętrznych serwerach, których utrzymaniem i ciągłością działania zajmuje się dostawca usług chmurowych. Zadaniami części chmurowej są:

* zapis danych, które przesyłane są z urządzeń podłączonych do Internetu;
* pośredniczenie w komunikacji między urządzeniami.

Drugi z wymienionych punktów jest rezultatem wymagania, aby końcówki systemu miały możliwość przesyłania między sobą informacji. Każde z urządzeń powinno posiadać pewien określony zbiór właściwości. Właściwości te, to konkretne cechy urządzenia opisujące jego stan. Istotą systemu jest możliwość łączenia poszczególnych właściwości różnych urządzeń, aby wpływały na siebie. Określenie tychże właściwości to część procesu modelowania urządzeń, czyli zbudowania listy wymagań na temat tego, jakie cechy powinny charakteryzować dany typ urządzenia. Więcej na temat modelowania można przeczytać w rozdziale „Modelowanie urządzeń” na stronie xx. Generalnie idea polega na tym, aby użytkownik miał możliwość tworzenia połączeń między wybranymi urządzeniami – w taki sposób, że jedno staje się sterownikiem, a inne aktuatorem. Ważną kwestią jest fakt, że połączenia takie powinny być możliwe do określenia w relacji wiele-do-wielu, to znaczy dowolna ilość urządzeń może sterować dowolną ilością innych urządzeń.

Opis przedstawiony na początku informuje, że każde z urządzeń powinno być oparte o mikrokontroler (wyposażony w łączoność wi-fi). Jednak w trakcie realizacji projektu, pomysł ten został rozszerzony – jako urządzenie rozumiany jest dowolny „element” będący w stanie podłączyć się do platformy. Może to więc być nie tylko mikrokontroler, ale także np. aplikacja komputerowa, która z wykorzystaniem pewnych bibliotek programistycznych może uzyskać dostęp do systemu. W pracy przyjęta została nomenklatura, według której każda „rzecz” podłączona do systemu jest urządzeniem – niezależnie od tego czy jest to fizyczny mikrokontroler, czy program komputerowy.

Użytkownicy systemu powinni mieć możliwość dokonywania jego konfiguracji – dotyczy to głównie definiowania połączeń między własnymi urządzeniami. W tym celu należało stworzyć aplikacje dostępowe, za pośrednictwem których użytkownik mógłby się zalogować z dowolnej platformy (PC (Windows/Linux), Mac, iOS, Android). Lepszym w utrzymaniu rozwiązaniem byłoby jednak utworzenie jednej aplikacji, która byłaby dostępna niezależnie od platformy jaką posiada użytkownik. Taka realizacja była możliwa dzięki obecności uniwersalnych i responsywnych technologii webowych.

Poszczególne części, które ogólnikowo zostały przedstawione w poszczególnych akapitach powyżej, zostaną znacznie bardziej szczegółowo opisane w następnych rozdziałach. Autor pracy traktował jej realizację jako okazję do poznania nowych narzędzi i zagadnień, w związku z czym starał się przedstawić proces tworzenia projektu z uwzględnieniem poczynionych obserwacji na ten temat.

Ze względu na fakt, że owocem pracy jest pewien produkt/usługa, należało go nazwać. W dalszej części pracy pojawiać się więc będzie określenie *MJIoT*, które autor wybrał jako wspomnianą nazwę. Pochodzi ono od inicjałów autora (MJ) oraz członu IoT.

## Przegląd rozwiązań obecnych na rynku

Zanim przejdę do opisu własnej implementacji systemu Internetu Rzeczy, warto poświęcić chwilę na zapoznanie się z rozwiązaniami, które są obecnie dostępne.

### IFTTT



Ilustracja 1.1 Logo usługi IFTTT

IFTTT (If This Then That) to platforma, która powstała w 2011 roku. Jej funkcjonalność polega na tworzeniu połączeń między wybranymi usługami przy wystąpieniu określonych warunków. Przykładowo, można zdefiniować połączenie między serwisem pogodowym a usługą mailową, w taki sposób, aby zawsze w przeddzień deszczowej pogody, użytkownik był o niej informowany. Siła rozwiązania polega na tym, że w tworzenie poszczególnych, dostępnych użytkownikowi, serwisów, zaangażowały się liczne firmy. Przykładem jest Google, które udostępnia usługi związane, między innymi, z Google Assisstant czy YouTube.

Jakiś czas temu, po zdobyciu popularności, do listy dostępnych serwisów dołączone zostały urządzenia (m. in. żarówki, przyciski). Pozwoliło to IFTTT stać się w wielu przypadkach rozwiązaniem z kategorii Smart Home, czyli automatyki domowej.

Omówione rozwiązanie było jedną z głównych inspiracji mojego projektu. Idea łączenia różnych usług/przedmiotów ze sobą, według upodobania użytkownika, jest bardzo ciekawym pomysłem. Pewnym minusem platformy jest jednak fakt, że jest to rozwiązanie zamknięte – użytkownik ma dostęp jedynie do serwisów i akcji, które przeszły proces certyfikacji IFTTT i zostały udostępnione globalnie. Tworzenie własnych urządzeń nie jest możliwe, zamiast tego należy kupić gotowe, dosyć drogie komponenty przygotowane przez firmy trzecie.

IFTTT jest niewątpliwie rozwiązaniem innowacyjnym i ciekawym. Jest to usługa komercyjna, która oferuje gotowe rozwiązania, bez możliwości tworzenia własnych komponentów systemu.

### Microsoft Flow



Ilustracja 1.2 Logo usługi Microsoft Flow

Powstały w 2016 roku serwis Flow jest niewątpliwie odpowiedzią firmy Microsoft na przedstawiony wcześniej IFTTT. Zasada działania projektu jest praktycznie ta sama, jednak dostępność usług, które można ze sobą łączyć jest, w porównaniu do IFTTT, dosyć niewielka. Microsoft Flow został umieszczony w niniejszym zestawieniu, aby pokazać, że koncepcja łączenia ze sobą różnych serwisów/produktów jest tematem, którym interesują się obecnie największe korporacje świata technologii.

Szukając informacji na temat podobnych rozwiązań okazuje się, że, oprócz opisanych, dostępnych jest wiele mniejszych rozwiązań, które bazują na tym samym schemacie, m. in.: Zapier, Automate.io, Huggin.

### Ubidots



Ilustracja 1.3 Logo platformy Ubidots

Kolejna z omawianych usług to, w przeciwieństwie do poprzednich, rozwiązanie kierowane głównie do przemysłu. System ten umożliwia kolekcję oraz wizualizację danych. Wyróżnia się mnogością dostępnych bibliotek programistycznych oraz łatwością dostosowania rozwiązania do swojej marki. Oprócz tego istnieje możliwość ustawienia powiadomień (mail, sms, itp.) w reakcji na określone zdarzenia.

Ważną kwestią w przypadku tego systemu jest jego cena. W przypadku potrzeby podłączenia do 60 urządzeń, koszt wynosi $99 miesięcznie – jest to usługa oferowana na zasadzie subskrypcji. Cena rośnie wraz z ilością urządzeń, które posiada klient.

Dwie pierwsze z przedstawionych ofert są bardzo podobne. Różnią się jednak znacznie od rozwiązania trzeciego (Ubidots), które skupia się na zbieraniu danych z podłączonych urządzeń i wizualizacji ich. Udowadnia to jedynie, że świat IoT jest bogaty w różne spojrzenia na to, czym tak naprawdę Internet Rzeczy jest. Wybrane rozwiązania nie zostały zaprezentowane przypadkowo. Projekt będący owocem niniejszej pracy stanowi próbę połączenia obu z przedstawionych scenariuszy:

* łączenie z sobą różnych „rzeczy”;
* gromadzenie danych z każdej z „rzeczy” i ich wizualizację.

# Usługi chmurowe

Istotnym jest by przybliżyć czytelnikowi czym są usługi chmurowe. Stanowią one dużą część projektu MJIoT, zostały zresztą wspomniane w samym tytule pracy. Chcąc omówić temat tego podrozdziału, najlepiej chyba odwołać się do definicji zagadnienia sformułowanej przez jednego z największych graczy na tym rynku – firmę Amazon:

Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing [4]

## Dostępne serwisy

Usługi chmurowe można więc najprościej określić jako zasoby informatyczne do wynajęcia. Zasoby te udostępniane są jako konkretne serwisy, najpopularniejsze z nich to np.:

* maszyny wirtualne,
* serwery baz danych bądź inne opcje przechowywania danych,
* pakiety biurowe,
* systemy wizualizacji danych.

Nie sposób wymienić wszystkich możliwych serwisów, gdyż każdy dostawca tego rodzaju rozwiązań stara się zaproponować coś, czego nie posiada konkurencja (bądź coś co byłoby lepsze). Obecnie mamy do czynienia z sytuacją, gdzie na rozwój chmur przeznaczane są ogromne inwestycje i w niedługich odstępach czasu pojawiają się kolejne nowości. Wystarczy np. śledzić wiadomości w kanale informacyjnym firmy Microsoft, by dowiedzieć się, że mniej więcej co kilka tygodni oferta platformy Azure (usługi chmurowej w ofercie Microsoftu) wzbogacana jest o nowe funkcjonalności.

Istotną cechą usług chmurowych jest metoda naliczania opłat. Zazwyczaj dostępne są dwie opcje:

* stała subskrypcja na określoną kwotę pieniężną, której nie można przekroczyć,
* tzw. Pay-As-You-Go, czyli płatność kwoty zgodnej z rzeczywistym użyciem.

Zaletami stosowania modelu chmurowego w stosunku do wcześniejszych są przede wszystkim:

* przeniesienie odpowiedzialności za utrzymanie infrastruktury na inny podmiot,
* możliwość większego skupienia się na budowie właściwego rozwiązania,
* łatwość skalowania rozwiązania w przypadku zmiany potrzeb na zasoby (również automatycznego, bez manualnej kontroli ze strony człowieka),
* dostępność ogromnej ilości serwisów w jednym miejscu – często z możliwością łatwej komunikacji między nimi.

Uważam, że ostatni punkt jest bardzo ważny. Dostawcy usług chmurowych, ze względu na silną konkurencję, starają się oferować swoje usługi w sposób, który jest jak najłatwiejszy do wykorzystania przez klienta. Oznacza to m. in. łatwość konfiguracji, która przekłada się bezpośrednio na mniejszy czas na nią poświęcony. Czas ten można przeznaczyć na dodatkowy rozwój własnego rozwiązania, co może wpłynąć na jego lepszą jakość czy konkurencyjność. Obecnie w informatyce mamy do czynienia z ogromną popularnością architektury mikro-serwisów, które idealnie wpisują się w warunki oferowane przez dostawców usług chmurowych. Izolacja poszczególnych komponentów aplikacji możliwa jest poprzez oddelegowanie oddzielnych zasobów/serwisów do obsługi konkretnego elementu. Połączenie tychże komponentów ze sobą jest znacznie ułatwione, jeśli znajdują się one na tej samej platformie. Często takie połączenia wymagają jedynie wybrania z listy konkretnych usług, które mają się komunikować – odpowiedzialność za zapewnienie poprawnej współpracy między nimi leży więc w dużej części po stronie dostawcy usługi, ponownie zwalniając konkretnego jej klienta z części obowiązków.

## Dostawcy usług chmurowych

Oferowanie usług chmurowych wiąże się z ogromnymi inwestycjami. W związku z tym nie dziwi, że najwięksi gracze w tej dziedzinie to firmy, które istniały często na długo zanim nastąpiła „rewolucja” chmurowa. Magazyn Forbes wylistował ich wg przychodów (dane za ostatni kwartał 2017 roku) [5]:

1. IBM Watson ($5,5B)
2. Microsoft Azure ($5,3B)
3. Amazon AWS ($5,1B)
4. Salesforce Cloud ($2,68B)
5. Oracle Cloud ($1,5B)
6. SAP ($1,24B)
7. Google Cloud ($1B)

Warto zastanowić się, czym wyróżniają się poszczególni dostawcy, aby być w stanie podjąć decyzję nt. platformy, o którą oparty zostanie projekt.

### IBM Watson



Ilustracja 2.1 Logo IBM Watson

Oferta korporacji IBM wyróżnia się przede wszystkim dużym naciskiem na uczenie maszynowe. Jest więc ona przeznaczona do rozwiązań, które mają bazować na „inteligentnym” operowaniu na danych wejściowych. Przykładowe wykorzystanie to m. in.:

* chatboty potrafiące przeprowadzić naturalną rozmowę z człowiekiem,
* analiza różnego rodzaju danych.

IBM Watson został przeze mnie odrzucony we wczesnym etapie planowania projektu ze względu na inny sposób rozumienia Internetu Rzeczy. Wizja IBM skupia się przede wszystkim na analizie danych z wykorzystaniem zaawansowanego uczenia maszynowego, co nie jest częścią mojego projektu. Ogromne zyski firmy IBM wskazują jednak, że przyjęła ona słuszny kierunek rozwoju swojej platformy, prawidłowo odczytując zapotrzebowania rynku.

### Microsoft Azure



Ilustracja 2.2 Logo Microsoft Azure

Firma Microsoft znana jest z wielu inicjatyw związanych z informatyką, nie mogło zabraknąć jej również w tym zestawieniu. Platforma Microsoft Azure jest obecnie źródłem największego dochodu firmy i stanowi cel jej największych inwestycji. Start usługi nastąpił w lutym 2010 roku. Spośród zalet tego usługodawcy można podkreślić:

* ogromną ilość oferowanych usług,
* gwarantowana dostępność na poziomie ok. 99,9% [6],
* dostępność interfejsów programistycznych dla wielu środowisk/języków, przede wszystkim: .NET, Python, Java, NodeJs,
* możliwość korzystania z platformy z opcją płatności Pay-As-You-Go,
* dostępność w ogromnej ilości regionów na całym świecie.

W przeciwieństwie do oferty IBM, propozycja Microsoftu zawiera niezbędne komponenty do realizacji projektu MJIoT. Ważną kwestią jest dostępność usług związanych z IoT (IoT Hub, który omówiony zostanie w dalszej części pracy) oraz dobre wsparcie dla programistów w postaci pakietów SDK (ang. *Software Develpment Kit*) oraz dokumentacji.

### Amazon AWS



Ilustracja 2.3 Logo Amazon AWS

Amazon, dawniej kojarzony głównie jako księgarnia internetowa, znajduje się bardzo wysoko na przedstawionej wcześniej liście. Wynika to przede wszystkim z faktu, że jest to najbardziej doświadczony z przedstawionych usługodawców – AWS (Amazon Web Services) pojawił się na rynku w marcu 2006 roku. Z tego powodu cieszy się sporym zaufaniem klientów, dla których liczy się „dojrzałość” wykorzystywanego środowiska.

Po krótkim omówieniu platformy Azure, ciężko tak naprawdę wskazać cechy odróżniające AWS - obie propozycje są do siebie dosyć podobne, głównie z powodu starań Microsoftu by dorównać Amazonowi. Spośród dostępnych serwisów, duża część z nich dostępna jest na obu platformach. Wydaje się, że obie nadają się do zastosowania w niniejszym projekcie. Należy jednak dokonać konkretnego wyboru, przyjrzyjmy się więc dokłądniej ofercie związanej z IoT w przypadku obu platform.

## PaaS, IaaS oraz SaaS

OMÓWIĆ

## Usługi IoT

Przy omawianiu usług IoT dostępnych na platformach chmurowych skupię się na ofercie firm Amazon (IoT Core) oraz Microsoft (IoT Hub). Pominę innych usługodawców, głownie ze względu na:

* mniejszą popularność (co przekłada się na mniejsze wsparcie),
* niedostępność stabilnej wersji usługi w trakcie prac nad projektem – np. Google opublikowało IoT Core dopiero w lutym 2018 roku,
* niską dostępność bibliotek programistycznych.

### Azure IoT Hub



Ilustracja 2.2 Logo Azure IoT Hub

IoT Hub to usługa, która publicznie wystartowała w kwietniu 2016 roku. Poniżej wylistowane zostały jej najważniejsze możliwości:

* komunikacja D2C (Device-to-Cloud) oraz C2D (Cloud-to-Device) z wykorzystaniem protokołów: MQTT, AMQP, HTTPS;
* retencja wiadomości, które nie mogły zostać dostarczone ze względu na status offline urządzenia (do 7 dni);
* autentyfikacja urządzeń;
* możliwość łączenia z wieloma innymi usługami, również z zastosowaniem własnych zasad biorących pod uwagę zawartość przesłanej wiadomości;
* wsparcie dla wielu środowisk programistycznych - C (w tym Arduino), .Net (Framework + Core), Python, NodeJS, Java, Swift, REST API.

### AWS IoT Core



Ilustracja 2.3 Logo AWS IoT Core

Data premiery AWS IoT Core to grudzień 2015 roku. Oto dostępne funkcjonalności:

* komunikacja urządzeń z chmurą oraz z innymi urządzeniami za pomocą protokołów: MQTT, HTTP, Websocket;
* autentyfikacja urządzeń;
* możliwość łączenia z innymi usługami AWS, z uwzględnieniem własnych zasad;
* wsparcie dla środowisk: C (w tym Arduino), JavaScript.

## Wybór dostawcy

Jak widać, obie firmy oferują dosyć podobne funkcjonalności. W swoim projekcie ostatecznie zastosowałem rozwiązania Microsoft (chmura Azure), głównie ze względu na:

* dobrą dokumentację z licznymi przykładami,
* bardzo dobrą dostępność pakietów SDK dla dużej ilości środowisk – w tym miejscu warto zwrócić szczególną uwagę na środowisko .NET, z którym autor pracy ma już pewne doświadczenie, a które zostało stworzone i jest silnie wspierane przez Microsoft,
* wcześniejszą znajomość platformy.

Przemyślany wybór docelowej platformy ma duże znaczenie również ze względu na koszty – korzystanie z zasobów chmurowych nie jest darmowe, warto więc dobrze się zastanowić, aby poświęcone czas i pieniądze nie poszły na marne.

# Modelowanie oraz łączenie urządzeń

Istotą działania systemu jest obsługa podłączonych do niego urządzeń – są one więc w centrum uwagi projektu. W związku z tym należało dobrze przemyśleć sposób ich definiowania i obsługi przez platformę. Oprócz tego należało się zastanowić jak tak naprawdę zrealizować łączenie urządzeń ze sobą. Czy zawsze jest to możliwe? Jakie dane powinny być przesyłane?

Informacje podane w niniejszym rozdziale zostaną rozszerzone w dalszej części pracy, szczególnie podczas omawiania bazy danych urządzeń.

## Definiowanie urządzeń

Każde urządzenie wewnątrz platformy MJIoT charakteryzowane jest przez dwa zbiory informacji:

* model określający zbiór informacji jaki powinien je reprezentować,
* instancję zawierającą informacje o konkretnym urządzeniu.

W celu lepszego zobrazowania zagadnienia, posłużę się przykładem prostego urządzenia – lampki elektrycznej. Zanim stworzymy konkretną lampkę (np. lampa w pokoju), musimy zdefiniować jego model – samą lampę w ogóle. Model ten powinien zawierać informacje określające zbiór właściwości jakie każda LAMPA będzie posiadać. W tym prostym przykładzie możemy przyjąć, że właściwości te przedstawiają się następująco:

* NAZWA – określa nazwę konkretnej lampy,
* STAN ŻARÓWKI – określa czy żarówka jest zapalona.

Nasz model posiada więc dwie właściwości. Na razie znamy jednak jedynie ich nazwy. Należy dodatkowo zdefiniować pewien zbiór meta-danych, które wzbogacą ich charakterystykę. Platforma pozwala określić następujące dane:

* Typ wartości – format jaki przyjmują wartości danej właściwości. Ma to znaczenie w momencie tworzenia połączenia między urządzeniami, co zostanie omówione w dalszej części pracy. Możliwe opcje to:
  + liczba (*numer*),
  + łańcuch tekstowy (*string*),
  + wartość zero-jedynkowa (*boolean*).
* Historyzowanie – określa czy dana właściwość powinna być przechowywana na zasadzie telemetrii, tzn. czy interesuje nas każda indywidualna jej wartość, która została zaraportowana przez urządzenie do chmury. Każda z tych wartości posiada dodatkowo informację o czasie jej przesłania (tzw. timestamp). Jeśli nie interesują nas wartości z przeszłości, przechowywana będzie jedynie ostatnia znana wartość. Parametr ten ma znaczenie ze względu na koszty przechowywania danych w chmurze;
* Zdolność sterowania – określa czy dana właściwość może być traktowana jako potencjalna wartość sterująca przy łączeniu z innym urządzeniem. W praktyce, parametr ten decyduje o tym, czy dane z pewnej konkretnej właściwości urządzenia mogą być przesyłane do innego urządzenia, by w jakiś sposób wpłynąć na jego stan;
* Zdolność bycia sterowanym – określa czy dana właściwość może być wykorzystywana w połączeniach jako odbiorca informacji z innych urządzeń.

Kontynuując przykład lamki, możemy wzbogacić informacje na temat modelu. Nasze dwie właściwości można opisać następująco:

* NAZWA:
  + Typ wartości: łańcuch tekstowy (*string*)
  + Historyzowanie – NIE
  + Zdolność sterowania – NIE
  + Zdolność bycia sterowanym – NIE
* STAN ŻARÓWKI:
  + Typ wartości: wartość zero-jedynkowa (*boolean*)
  + Historyzowanie – TAK
  + Zdolność sterowania – NIE
  + Zdolność bycia sterowanym – TAK

Pozostają jeszcze jedynie dwa parametry dotyczące modelu:

* Abstrakcyjność – określa czy na podstawie danego modelu można tworzyć instancje;
* Aktywna komunikacja offline – określa czy do urządzenia, będącego instancją danego modelu, powinny być przesyłane komunikaty, kiedy nie ma ono połączenia z platformą.

Znaczenie obu parametrów zostanie wyjaśnione w dalszej części pracy. Na razie załóżmy, że nasz model lampki:

* nie jest abstrakcyjny,
* nie akceptuje wiadomości będąc offline.

Dopiero mając definicję modelu możemy przystąpić do stworzenia w bazie platformy konkretnej jego instancji. Przykładowo:

Lampa w pokoju:

* NAZWA: „Lampa w pokoju”;
* STAN ŻAROWKI: domyślnie wyłączona (wartość false).

Oprócz tego należy również zdefiniować:

* do kogo należy urządzenie (numer identyfikacyjny użytkownika),
* jaki jest klucz urządzenia do połączenia z IoT Hub’em.

Określony powyżej zbiór informacji jest przydatny na różnych poziomach działania platformy, zarówno w części „wewnętrznej”, niewidocznej dla użytkownika, jak i w części udostępnionej publicznie – w interfejsie aplikacji klienckiej.

## Dziedziczenie właściwości

Przedstawiony proces tworzenia modeli jest wystarczający by utworzyć wiele typów urządzeń wewnątrz platformy. W niektórych przypadkach może się zdarzyć, że nowozdefiniowany model jest bardzo podobny do encji, która została już wcześniej utworzona. W takim przypadku możliwe jest dziedziczenie właściwości istniejącego modelu, aby wykorzystać istniejącą konfigurację i ją rozbudować. Przykładem może być tutaj urządzenie typu pilot. Przyjmijmy, że posiadamy zdefiniowany w bazie model PILOT10, który posiada 10 przycisków. Każdy przycisk reprezentowany jest przez właściwość o nazwie „Stan przycisku N”, gdzie N oznacza numer przycisku. W pewnym momencie jednak chcielibyśmy dodać do platformy inny rodzaj pilota – taki, który posiada 15 przycisków. Moglibyśmy od nowa definiować każdy z 15 przycisków nowego pilota. Lepszym rozwiązaniem będzie jednak wykorzystanie istniejących 10 właściwości, które stworzyliśmy na potrzeby modelu PILOT10 i dodać brakujące 5 – w taki sposób powstanie nowy model – PILOT15.

Dziedziczenie modeli nie jest mechanizmem niezbędnym do działania platformy. Mimo to jest to ciekawa funkcjonalność, która może się przydać przy definiowaniu wielu typów urządzeń o podobnych właściwościach.

Zdefiniowana w poprzednim podrozdziale LAMPA jest bardzo prostym modelem. Również w jej przypadku jednak możemy zastosować dziedziczenie – dotyczy to właściwości NAZWA. Bez względu na to czym jest dane urządzenie, powinno ono posiadać jakąś nazwę, dzięki której mogłoby być szybko kojarzone przez użytkownika podczas zarządzania nim z poziomu aplikacji klienckiej. Jest to więc dobra okazja do tego, by utworzyć model bazowy – nazwiemy go MODEL\_BAZOWY (wewnątrz platformy MJIoT został on nazwany BaseModel). Jedyną właściwością, którą posiada jest NAZWA, a jej charakterystyka jest taka sama jak przedstawiona wcześniej podczas omawianiu przykładu modelu lampy.

MODEL\_BAZOWY różni się od modeli LAMPA czy PILOT10 pewną istotną cechą – nigdy nie stworzymy jego instancji, gdyż jest to typ stworzony jedynie na potrzeby dziedziczenia. W związku z tym w bazie danych zostanie on oznaczony jako model abstrakcyjny.

Opisany mechanizm umożliwia przejmowanie właściwości po tylko jednym typie. Nie ma jednak przeciwskazań, a nawet wydaje się to dobrym pomysłem, aby umożliwić dziedziczenie z wielu modeli. W ten sposób jeszcze bardziej przyspieszyć można by tworzenie nowych typów urządzeń.

Przedstawiony sposób definicji modeli oraz instancji został zainspirowany podejściem stosowanym w programowaniu obiektowym, gdzie definiujemy klasy oraz obiekty (instancje klas). Pokazać to można, wracając na chwilę do przykładu modelu LAMPA i definiując go z wykorzystaniem składni obiektowego języka programowania, np. C#:

//Nie można utworzyć instancji klasy ModelBazowy

public abstract class ModelBazowy

{

public string Nazwa { get; set; }

}

//Lampa dziedziczy z ModelBazowy

public class Lampa : ModelBazowy

{

public bool StanZarowki { get; set; }

}

…

//Konkretna instancja klasy Lampa

var lampkaWPokoju = new Lampa {

Nazwa = "Lampa w pokoju",

StanZarowki = false

};

Jak zostanie pokazane w rozdziale omawiającym bazę danych urządzeń, obiektowe podejście zostało zastosowane w praktyce, za sprawą zastosowania systemu ORM do budowy struktury bazy i wydobywania z niej informacji.

Wspomniane zostało, że proces modelowania mógłby zostać rozbudowany o możliwość wielokrotnego dziedziczenia. Inną funkcjonalnością, która wydaje się być jeszcze bardziej istotna to możliwość definiowania metod dla urządzeń. Bardziej naturalnym byłoby np. wydać polecenie „włącz żarówkę” niż ustawienie właściwości STAN\_ŻARÓWKI na wartość TRUE. Niestety jednak autor projektu nie przewidział takiej możliwości podczas jego planowania. Wskazuje to jedynie na fakt, że etap definiowania założeń jakiegokolwiek projektu jest bardzo istotny i należy mu poświęcić wiele uwagi. W pewnym momencie jest już zbyt późno by wprowadzać do projektu nieplanowane wcześniej funkcjonalności bez istotnego zaburzenia zaplanowanego terminu jego ukończenia.

## Łączenie urządzeń

Zagadnienie, któremu niewątpliwie należy poświęcić uwagę jest sposób łączenia urządzeń ze sobą. W podrozdziale „Definiowanie urządzeń” wspomniane zostały dwa parametry, które posiada każda właściwość modelu urządzenia:

* Zdolność sterowania,
* Zdolność bycia sterowanym.

Są to kluczowe parametry do wyjaśnienia tematu tego podrozdziału. Autor pracy zastosował koncepcję według której łączone są ze sobą nie same urządzenia, lecz ich właściwości. Dla przykładu, przyjmijmy, że posiadamy urządzenia: przycisk oraz lampka, o następującym zestawie właściwości:

PRZYCISK:

* NAZWA (łańcuch tekstowy)
* STAN PRZYCISKU (wartość zero-jedynkowa, posiada zdolność sterowania)

LAMPKA:

* NAZWA (łańcuch tekstowy)
* STAN ŻARÓWKI (wartość zero-jedynkowa, posiada zdolność bycia sterowanym)
* KOLOR ŚWIECENIA (łańcuch tekstowy, posiada zdolność bycia sterowanym)

Chcąc je ze sobą połączyć, nie możemy po prostu zdefiniować relacji w sposób następujący:

Połącz urządzenie PRZYCISK z urządzeniem LAMPKA

Taka definicja nie daje wystarczających informacji o tym, czego tak naprawdę oczekujemy jako użytkownicy platformy. PRZYCISK posiada tylko jedną właściwość sterującą, lecz LAMPKA wyposażona jest w dwie właściwości sterowalne. System musi wiedzieć dokładnie, która z nich ma zostać wybrana jako sterowana.

Zamiast tego połączenie powinno być więc zdefiniowane np. w poniższy sposób:

STAN PRZYCIKU urządzenia PRZYCIK ma wpływać na STAN ŻARÓWKI urządzenia LAMPKA.

Taka definicja opisuje już znacznie lepiej, co chcemy osiągnąć. Określone zostały konkretne właściwości obu urządzeń, które mają zostać połączone. Po dodaniu takiej definicji połączenia do bazy danych, otrzymamy następujące zachowanie lampki:

* przy wciśnięciu przycisku – lampka zapali się;
* przy wyłączeniu przycisku – lampka zostanie zgaszona.

STAN ŻARÓWKI odwzorowuje więc STAN PRZYCISKU. W wielu przypadkach taki charakter relacji PRZYCISK-LAMPKA jest oczekiwany i wystarczający. Co jednak w sytuacji, kiedy chcielibyśmy np. odwrócić reakcję lampki, tzn.:

* przy wciśnięciu przycisku – lampka ma zostać zgaszona;
* przy wyłączeniu przycisku – lampka ma się zapalić.

Rozwiązanie takiego scenariusza (oraz innych) zostało zrealizowane za pomocą **kalkulacji**.

### Kalkulacje

Sytuacja przedstawiona pod koniec ostatniego podrozdziału jest jednym z wielu możliwych przypadków, kiedy użytkownik chciałby wpłynąć na działanie zdefiniowanego połączenia. Inne przykładowe możliwości to:

* zapalanie lampki za każdym razem, niezależnie od stanu samego przycisku
* gaszenie lampki za każdym razem, niezależnie od stanu samego przycisku.

Każda z przedstawionych sytuacji wymaga wprowadzenia mechanizmu, który byłby w stanie

# Omówienie projektu

Wiedząc czym jest Internet Rzeczy oraz usługi chmurowe i jakie możliwości oferują, możemy przejść do opisu realizacji projektu, któremu poświęcona jest niniejsza praca. W tym rozdziale zaprezentowana zostanie lista komponentów z jakich składa się system, wraz ze schematem, który prezentuje architekturę całej platformy.

## Schemat platformy



Ilustracja 3.1 Schemat platformy MJIoT

Powyższy schemat jest uproszczonym, ogólnym spojrzeniem na platformę. Strzałki odchodzące od poszczególnych elementów wskazują komponenty, z których dany element korzysta w celu wykonania swojej funkcji. Wspomniane uproszczenie schematu wynika z faktu, że każdy ze zilustrowanych komponentów można podzielić na mniejsze części, które zostaną omówione w odpowiednich podrozdziałach.

Rysunek możemy go podzielić na trzy części:

* komponenty chmurowe, które stanowią wewnętrzną część platformy i pozostają „ukryte” przez użytkownikiem;
* komponenty, które znajdują się w chmurze, lecz stanowią punkt dostępowy dla użytkowników systemu (są dla nich widoczne);
* urządzenia, które dany użytkownik posiada.

## Komponenty umieszczone w chmurze

Pierwsza z wymienionych kategorii zawiera elementy, które tworzą sam fundament platformy, dzięki którym możliwe jest jej funkcjonowanie. Oto ich lista:

* IoT Hub – przekazywanie wiadomości między chmurą a urządzeniami;
* Baza urządzeń – baza danych, która zawiera informacje na temat:
  + istniejących modeli urządzeń,
  + istniejących instancji urządzeń,
  + połączeń między urządzeniami,
  + użytkowników zarejestrowanych w systemie.
* Baza telemetrii – baza danych, która gromadzi wszelkie informacji przesyłane z urządzeń do chmury w celu ich odczytu bądź wizualizacji;
* Logika systemu – programy, które wykonują odpowiednie akcje przy otrzymaniu komunikatu z urządzenia:
  + wysyłanie sygnałów sterujących do połączonych urządzeń,
  + zapis danych telemetrycznych do bazy danych.

Z punktu widzenia użytkownika, nie potrzebuje on żadnej informacji na temat tego jakie są to komponenty. Co więcej, mogłyby one zostać przebudowane bądź kompletnie wymienione na inny zestaw serwisów, czego użytkownik nie powinien w ogóle zauważyć. Ponownie odwołam się do zasad znanych z programowania obiektowego – tworząc klasę, udostępniamy na zewnątrz jedynie część publiczną jej funkcjonalności. Część prywatna pozostaje ukryta, chociaż publiczny interfejs korzysta z niej w celu wykonania odpowiednich funkcji. Dokładnie na tej samej zasadzie zrealizowany został projekt MJIoT.

### IoT Hub

Usługa IoT Hub została już przedstawiona w jednym z wcześniejszych rozdziałów. W tej części pracy omówione zostanie jednak jej praktyczne wykorzystanie w projekcie. Zacząć należy od samej czynności dodania usługi do swojego konta. W trakcie tego procesu określić należy kilka ustawień, które w tym przypadku są uniwersalne względem praktycznie wszystkich usług platformy Azure. Są to m. in.:

* region usługi,
* próg cenowy
* grupa zasobów

Przy ustawianiu pierwszej z wymienionych opcji należy zastanowić się, gdzie usługa będzie wykorzystywana, gdyż nieodpowiedni wybór regionu może spowolnić jej działanie. W związku z tym w przypadku MJIoT wybór padł na region *West Europe*. Generalnie jednak, wybór regionu może mieć w niektórych przypadkach większe znaczenie. Są ku temu dwa powody:

* w niektórych regionach niedostępne są niektóre funkcjonalności Azure. Przykładowo w przypadku regionu West Europe, usługa IoT Hub posiada dezaktywowaną funkcję „Manual failover” (zabezpieczenie przed katastrofami w regionach, nieistotne w niniejszym projekcie):



Ilustracja 4.1 Komunikat nt. braku dostępu do funkcjonalności "Manual failover" w regionie West Europe

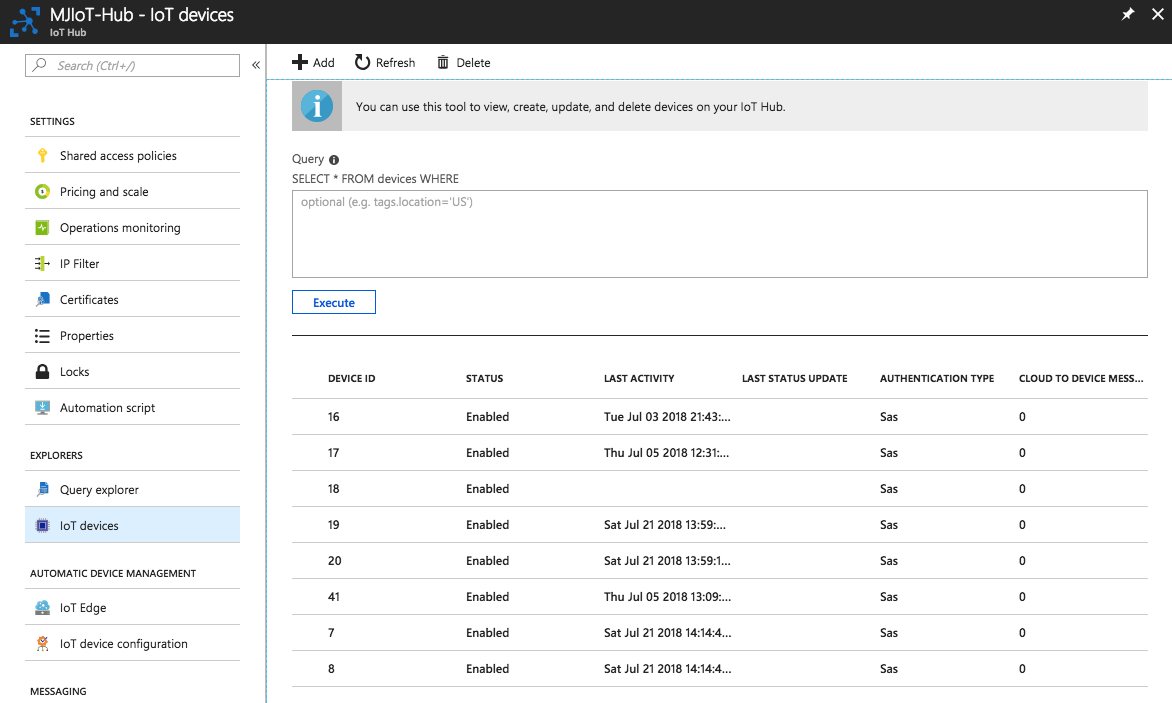
* względy geopolityczne – niektóre państwa (np. Niemcy oraz Chiny) legislacyjnie zabraniają przechowywania danych procesowych swoich przedsiębiorstw poza granicami kraju.

Kolejny z punktów – próg cenowy – określa ilość dostępnych zasobów oraz przewidywaną kwotę pieniężną, jaką właściciel subskrypcji będzie musiał za nią zapłacić. W przypadku serwisu IoT Hub dostępnych jest siedem progów cenowych, a różnią się one maksymalną ilością wiadomości jakie będą mogły być wysłane poprzez serwis w ciągu dnia. W tym przypadku dostępny jest próg F1, który jest darmowy i oferuje 8000 wiadomości w ciągu dnia. Jest to ilość niewielka, szczególnie w trakcie rozwoju projektu.



Ilustracja 4.2 Wybór progu cenowego usługi IoT Hub

Po stworzeniu instancji usługi na swoim koncie, mamy dostęp do panelu konfiguracyjnego:



Ilustracja 4.3 Lista zarejestrowanych urządzeń jako jedno z wielu dostępnych okien w panelu IoT Hub

Istotnym punktem, jeśli chodzi o konfigurację, są tzw. grupy konsumentów. Mają one znaczenie w przypadku, kiedy wiadomości odbierane są przez więcej niż 1 odbiorcę. Taka sytuacja zachodzi ze względu na zastosowanie w projekcie architektury mikroserwisów. Platforma zawiera 2 serwisy nasłuchujące:

* odpowiedzialny za obsługę połączeń między urządzeniami;
* odpowiedzialny za zapis telemetrii.

Okazuje się, że jedna grupa może tylko raz odebrać jedną wiadomość wysłaną przez IoT Hub. W przypadku, kiedy oba z wymienionych serwisów korzystały z tej samej grupy, pojedynczy komunikat był procesowany przez tylko jeden z tych serwisów. Przez pewien czas nie byłem w stanie zdiagnozować istoty problemu i podejrzewałem, że wada występuje w napisanym przeze mnie kodzie logiki platformy, podczas gdy w rzeczywistości należało stworzyć dodatkową grupę.



Ilustracja 4.4 Grupy konsumentów usługi IoT Hub zdefiniowane na potrzeby serwisów nasłuchujących

IoT Hub to jeden z najważniejszych elementów systemu. Jest tak, ponieważ to za jego pośrednictwem urządzenia mogą nawiązywać komunikację z chmurą. Każda „rzecz” będąca podłączona do platformy posiada zależności w postaci bibliotek programistycznych IoT Hub. W projekcie wykorzystane zostały biblioteki:

* *Microsoft.Azure.Devices.Client* (.NET) – pozwala m. in. wysyłać komunikaty do chmury z poziomu aplikacji napisanej w platformie .NET (np. z wykorzystaniem języka C#);
* *Microsoft.Azure.Devices* (.NET) – zastosowana wewnątrz logiki platformy w chmurze, pozwala m. in. przesyłać komunikaty do urządzeń;
* *Azure IoT Device SDK* (C) – pozwala podłączyć do IoT Hub’a urządzenia z kategorii mikrokontrolerów.

Dla środowiska .NET, na potrzeby niniejszego projektu, utworzona została dodatkowo biblioteka *MjIot.Devices.Common*, która udostępnia uproszczony zestaw API do obsługi urządzeń. Szczegóły omówione zostaną w rozdziale dotyczącym urządzeń.

## Baza urządzeń Azure SQL

W jednym z poprzednich rozdziałów omówione zostało modelowanie urządzeń. Dane uzyskane w tym procesie należy gdzieś przechować – dla tego celu wykorzystana została kolejna usługa udostępniana w chmurze Azure – relacyjna baza danych Azure SQL. Jest to baza stworzona w oparciu o oprogramowanie Microsoft SQL Server. Obecnie jednak systemy te rozwijane są osobno i każdy z nich posiada pewien oddzielny zbiór funkcjonalności. W przypadku bazy Azure SQL chodzi tu przede wszystkim o zdolność do skalowania zasobów w ramach potrzeb – jest to cecha wspólna usług dostępnych w chmurze.

### Skalowanie serwisów

W tradycyjnym modelu publikowania rozwiązania informatycznego, osoba odpowiedzialna za wdrożenia miała dwie opcje [7]:

* przydzielić zasoby w oparciu o wykorzystanie szczytowe produktu,
* przydzielić zasoby pokrywające zwyczajne/najczęstsze zapotrzebowania.

Oba rozwiązania mają swoje wady. Pierwsze wiąże się z nadmiernymi kosztami w trakcie, kiedy dodatkowe zasoby nie są tak naprawdę wykorzystane. Druga opcja natomiast wiąże się z ryzykiem utraty klientów, którzy nie będą w stanie skorzystać z naszego produktu z powodu przeciążenia serwerów. Przedstawiona sytuacja jest jednym z głównych powodów atrakcyjności rozwiązań chmurowych. Posiadają one mechanizmy, które w prosty sposób pozwalają skonfigurować mechanizmy skalowania. Można tu wymienić dwa rodzaje:

* skalowanie pionowe (ang. *scaling-up*) – polega na zwiększeniu zasobów maszyny obsługującej dany serwis, co najczęściej oznacza zastosowanie maszyny o większej ilości pamięci oraz dodatkowej mocy obliczeniowej;
* skalowanie poziomowe (ang. *scaling-out*) – polega na rozdzieleniu pracy serwisu na wiele serwerów.

Projekt MJIoT nie korzysta z mechanizmów skalowania ze względu na koszty takiego rozwiązania. Warto było jednak wspomnieć o takiej możliwości, ponieważ jest to ważna funkcjonalność udostępniana przez dostawców usług chmurowych. Projekty oparte o chmurę zdecydowanie mogą zyskać na zastosowaniu tego podejścia. Również niniejszy projekt prawdopodobnie korzystałby ze skalowania, gdyby był wykorzystywany na szerszą skalę.

### Struktura bazy danych

Baza danych typu SQL składa się z tabel, które połączone są często pewnymi relacjami. Dobrym sposobem przedstawienia takiej struktury jest diagram UML:

WSTAWIĆ DIAGRAM UML BAZY SQL

Baza urządzeń zawiera 5 tabel:

* DeviceTypes,
* PropertyTypes,
* Devices,
* Connections,
* Users.

Pierwsze dwie z wymienionych tabel służą do przechowywania informacji o zdefiniowanych modelach, a ich kolumny odpowiadają opisowi przedstawionemu w rozdziale „Modelowanie oraz łączenie urządzeń”.

# Bibliografia

[1] <https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf>

[2] “The Internet of Things: In a Connected World of Smart Objects” (Accenture & Bankinter Foundation of Innovation, 2011)

[3] HP Security, Miessler, 2014

[4] <https://aws.amazon.com/what-is-cloud-computing/>

[5] <https://www.forbes.com/sites/bobevans1/2018/04/09/microsoft-amazon-and-ibm-which-cloud-powerhouse-will-top-q1-revenue-charts/#2872148814dc>

[6] <https://azure.microsoft.com/en-us/support/legal/sla/summary/>

[7] https://docs.microsoft.com/pl-pl/azure/sql-database/sql-database-elastic-pool