**POLITECHNIKA CZĘSTOCHOWSKA**

**WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI**



PRACA DYPLOMOWA INŻYNIERSKA

**System zdalnego monitorowania stanu i sterowania urządzeń.**

*System remote monitoring of the status and control devices.*

Michał Walaszczyk

Nr albumu: 116066

Kierunek: informatyka

Studia: stacjonarne

Poziom studiów: pierwszy

Promotor pracy: dr inż. Jerzy Jelonkiewicz

Praca przyjęta dnia:

Podpis promotora:

Częstochowa, 2015**Spis treści**

[Cel i zakres pracy 3](#_Toc440245858)

[Wstęp 4](#_Toc440245859)

[1 Internet rzeczy 6](#_Toc440245860)

[2 Urządzenia 7](#_Toc440245861)

[2.1 Raspberry Pi 7](#_Toc440245862)

[2.2 Sense Hat 10](#_Toc440245863)

[3 Oprogramowanie 11](#_Toc440245864)

[3.1 Windows 10 IoT Core 11](#_Toc440245865)

[3.2 Visual Studio 13](#_Toc440245866)

[3.3 Microsoft Azure 13](#_Toc440245867)

[4 Komunikacja 15](#_Toc440245868)

[4.1 Advanced Message Queuing Protocol 15](#_Toc440245869)

[4.2 Hypertext Transfer Protocol 16](#_Toc440245870)

[5 Realizacja 17](#_Toc440245871)

[5.1 Instalacja Windows 10 IoT Core 17](#_Toc440245872)

[5.2 Utworzenie IoT Hub 18](#_Toc440245873)

[5.3 Rejestrowanie urządzenia 19](#_Toc440245874)

[5.4 Przesyłanie komunikatów z urządzenia do chmury 21](#_Toc440245875)

[5.5 Odbieranie komunikatów od urządzeń 23](#_Toc440245876)

[Podsumowanie 25](#_Toc440245877)

[Literatura 26](#_Toc440245878)

[Dodatek A. Spis zawartości dołączonej płyty CD 27](#_Toc440245879)

Cel i zakres pracy

* 1. Celem niniejszej pracy jest zaprojektowanie systemu, dzięki któremu będziemy mogli zdalnie monitorować stan urządzeń oraz sterować nimi. Dziedzina zdalnej kontroli nad dowolnymi urządzeniami jest bardzo szeroka i wybranie odpowiedniej metody komunikowania się z urządzeniami może stanowić problem. Przy wyborze należy wziąć pod uwagę wiele aspektów, aby metoda była odpowiednia do danego systemu.
     1. W swojej pracy chcę podjąć się zaprojektowania systemu zdalnej kontroli dowolnych urządzeń podłączonych do platformy komputerowej Raspberry Pi przez użytkownika, który będzie miał dostęp do nich przez dedykowaną aplikację z dowolnego miejsca na Ziemi. I tutaj nasuwa się pytanie czy przypadkiem takie rozwiązania nie były już podejmowane przez wiele osób? Zgadza się. Lecz większość tych rozwiązań nie przewiduje skalowalności swojego systemu. Wyobraźmy sobie, że stworzyliśmy jakieś rozwiązanie i chcemy udostępnić je szerszemu gronu odbiorców. Chcemy, aby każdy z użytkowników mógł podglądać parametry pracy swojego urządzenia lub sterować pracą jego urządzeń z jednego portalu WWW dostępnego dla wszystkich użytkowników. Pojawia się problem identyfikowania urządzenia do danego użytkownika systemu tak, aby nad jego urządzeniami mógł mieć kontrole tylko on, a komunikacja z nim odbywała się szybko, bezpiecznie i bezawaryjnie. Dlatego też praca ta będzie w głównej mierze pokazywała rozwiązanie problemu skalowalności systemów sterowania, czyli wybranie odpowiedniej metody komunikowania się z urządzeniem, która będzie spełniać te warunki.
     2. Ponieważ informatyka jest dziedziną bardzo dynamicznie rozwijającą się to chcę na tyle na ile to możliwe wykorzystać najnowsze technologie obecne na rynku. Jest to bardzo ważne dla każdego programisty, aby w dzisiejszych czasach wykorzystywać i znać nowości, które oferują nam najwięksi dostawcy technologii tacy jak np. Microsoft. Dlatego tez praca będzie obejmować opis wszystkich technologii i rozwiązań wykorzystanych w projekcie. Pomoże to dokładnie zgłębić temat i poszerzyć swoją wiedzę w tym zakresie. Oprócz tego chcę również pokazać praktyczne wykorzystanie ich w swoim projekcie, aby udowodnić, że ich wykorzystanie jest rozwiązaniem problemu odpowiedniej komunikacji w świecie internetu rzeczy.

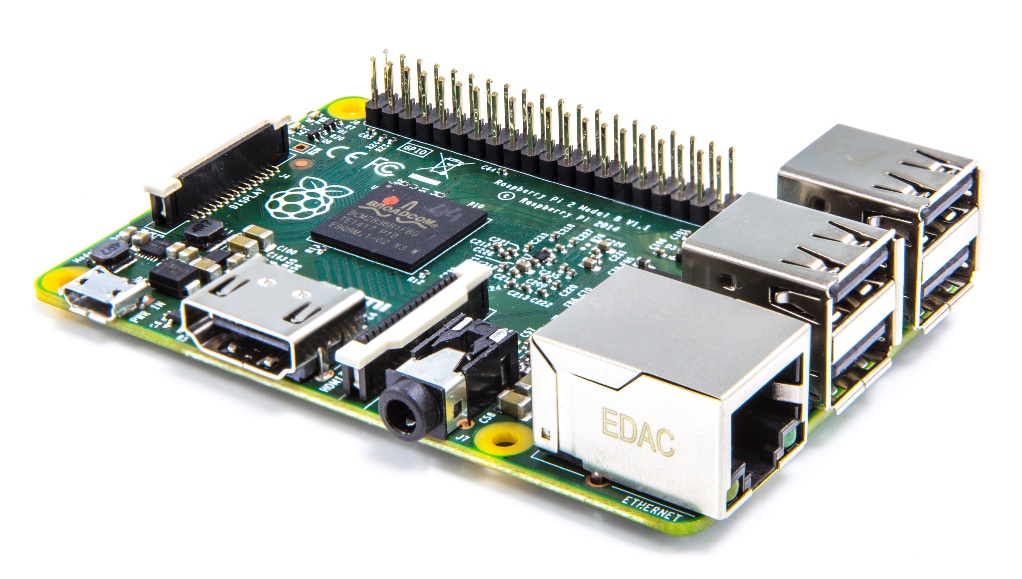
Wstęp

* + 1. W ostatnim czasie możemy zaobserwować intensywny rozwój dziedziny informatyki, która zajmuje się systemami wbudowanymi. W naszym codziennym życiu pojawia się coraz więcej elektroniki, od której jesteśmy coraz bardziej uzależnieni. Nie da się ukryć, że oprócz już istniejących urządzeń elektronicznych, które działają pod kontrolą jakiegoś systemu, producenci różnych przedmiotów chcą, aby również w ich produktach pojawiła się elektronika. Jednak nie poprzestają tylko i wyłącznie na zwykłym systemie sterującym. Chcą, aby ich produkty łączyły się do Internetu. Daje im to możliwość zbierania informacji o pracy każdego pojedynczego wyprodukowanego urządzenia, ale i również zdalną kontrolę nad nim nie tylko przez producenta, ale i użytkowników docelowych. Również w fabrykach coraz więcej procesów produkcyjnych analizuje się poprzez zbieranie danych z linii produkcyjnych, na których rozmiesza się równego rodzaju urządzenia rejestrujące. Taka wizja przyszłości gdzie z praktycznie każdym urządzeniem będziemy mogli połączyć się za pośrednictwem internetu jest poruszana przez naukowców już od kilkunastu lat i nazywa się „internet rzeczy”. Czy wydaje się ona realna? Myślę, że jak najbardziej. Przykładem realizacji dążenia do tej koncepcji może być chociażby szczoteczka do zębów firmy Oral-B, która rejestruję proces mycia zębów i poprzez aplikację pokazuje użytkownikowi czy dobrze wykonuje czynności prawidłowego mycia zębów. Patrząc na ten przykład nie mam wątpliwości, co do przyszłości rozwoju tej dziedziny informatyki. Wątpliwości nie mają także największe koncerny informatyczne, które zauważyły bardzo duży wzrost liczby takich urządzeń na rynku. Wiedząc jak duża jest ta strefa runku i jak duże pieniądze będzie można w przyszłości z niej pozyskać wychodzą na przód oczekiwaniom twórców i tworzą swoje rozwiązania mające na celu polepszenie pracy takich urządzeń, ale i także kształcenia nowych pokoleń w kierunku tworzenia takich systemów. Stąd pojawiło się bardzo dużo możliwości nauki w kierunku systemów wbudowanych. Mamy do wyboru wiele platform programistycznych między innymi Raspberry, Arduino czy Galileo i wiele innych, które dają duże możliwości nauki za niewielką cenę tych produktów. Pojawienie się tak wielu firm zainteresowanych tą tematyką znacznie przyśpieszyło rozwój tej dziedziny informatyki. Lecz wraz z rosnącą ilością sprzętu zaczęły pojawiać się dylematy związane z oprogramowaniem. Do niedawna mieliśmy do wyboru systemy bazujące na jądrze Linuxa, lecz swój wkład w rozwój tego nurtu postanowił włożyć Microsoft tworząc wersję swojego najnowszego systemu operacyjnego przystosowaną na urządzenia wbudowane. Nie poprzestał jednak na tylko na tym. Microsoft zauważył również, że problemem w świecie urządzeń internetu rzeczy jest komunikacja. Ponieważ ilość urządzeń rośnie w bardzo szybkim tempie to efektywny sposób zbierania informacji z każdego urządzenia może stanowić problem. Dochodzi ona czasem nawet od kilkuset tysięcy do miliona. Producenci rozwiązań oczekują sprawnego i szybkiego zbierania informacji z urządzeń. Dlatego też Microsoft postanowił do swojej oferty platformy chmurowej Azure dodać nową usługę, która została stworzona specjalnie pod urządzenia internetu rzeczy. Wykorzystuje ona komunikację przy pomocy protokołów warstwy pośredniczącej. Jest to dość powszechnie stosowane rozwiązanie między innymi w bankowości, czy najpopularniejszej aplikacji Messenger od Facebooka. W związku z tym ciekawym wydaje się wykorzystanie takiego modelu komunikacji z urządzeniami internetu rzeczy i sprawdzenie czy takie rozwiązanie może sprawdzić w rzeczywistości. Przy okazji mamy okazję zaznajomić się z działaniem platformy chmurowej, która ostatnimi czasy staje się bardzo popularna. Na rynku pojawia się coraz więcej aplikacji współpracujących z platformami chmurowymi. Możemy dostrzec bardzo wiele zalet takich rozwiązań. Między innymi odciążenie mniejszych elementów naszego systemu i ograniczenie zakresu ich pracy tylko i wyłączenie do obsługi czujników, a cała reszta, czyli obliczenia, które mogą być bardzo obciążające i analiza otrzymywanych danych telemetrycznych może odbywać się w innym miejscu gdzie mamy dostęp do odpowiednich zasobów obliczeniowych. Komunikatami możemy również sterować urządzeniami wysyłając do nich i odpowiednio przetworzone komunikaty, które będą w stanie rozpoznać. A także nie mamy ograniczenia, co do zawartości tych wiadomości gdyż są one przesyłane w postaci binarnej tak, więc wszystkie dane, które jesteśmy w stanie serializować do strumienia bajtów możemy przesłać do chmury lub urządzenia.

1. Internet rzeczy

* 1. Internet rzeczy (and. *Internet of Things – IoT*) to koncepcja sieci fizycznych obiektów lub rzeczy zbudowanych z elektroniki, oprogramowania, sensorów i łącza sieciowego, co umożliwia im zbieranie i wymianę danych. Internet rzeczy pozwala obiektom być wykrywanym i zdalnie sterowanym poprzez istniejącą infrastrukturę sieciową, tworzenie możliwości bardziej bezpośredniej integracji świata fizycznego i systemów komputerowych, a także poprawy wydajności, dokładności i oszczędności finansowych. Każda rzecz jest jednoznacznie zidentyfikowana poprzez wbudowany system komputerowy, ale jest w stanie współpracować w ramach istniejącej infrastruktury internetowej. Eksperci szacują, że Internet przedmiotów będzie się składać z 50mld obiektów do 2020 roku. Termin internetu rzeczy po raz pierwszy pojawił się w 1999 roku. Wymyślił go brytyjski przedsiębiorca Kevin Asthon. Oczekuje się, że IoT oferuje zaawansowaną łączność z urządzeniami, systemami i usługami, która wykracza poza komunikację maszyna-do-maszyny i obejmuje wiele protokołów, domen i aplikacji. Wzajemne połączenie urządzeń wbudowanych (w tym inteligentnych obiektów) dąży do wprowadzenia automatyzacji w praktycznie wszystkich dziedzinach życia i rozszerzanie się obszarów między innymi takich jak nie tylko inteligentne domy, ale i inteligentne miasta. „Rzeczy” w kontekście internetu rzeczy może odnosić się do wielu różnych urządzeń, na przykład specjalistycznych takich jak implanty monitorujących serce, auta z wbudowanymi czujnikami lub urządzeń wspomagających akcje ratownicze lub poszukiwawcze, ale również proste urządzenia gospodarstwa domowego takie jak pralki, grzejniki, lampy czy liczniki prądu/wody. Podsumowując, internet rzeczy to jakakolwiek rzecz podłączona do sieci, która ma zdolność do komunikowania się ze światem zewnętrznym, aby ułatwić ludziom codzienne funkcjonowanie.

1. Urządzenia
   1. W niniejszym rozdziale przedstawiony jest sprzęt elektroniczny wykorzystany w pracy, który posłużył do testów, jako urządzenie, z którym możemy się komunikować. Lista nie jest długa, ponieważ nie chciałem skupiać się na praktycznych możliwościach komunikacji z czujnikami, a jedynie na gotowym rozwiązaniu (kompletnym urządzeniu) z chmurą. Dlatego też wykorzystane komponenty są łatwe w obsłudze i złożeniu oraz posiadają bardzo dużo funkcjonalności jak na tak małe urządzenia.
      * 1. C:\Users\Michał\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Raspberry_Pi_Logo.svg.pngRaspberry Pi

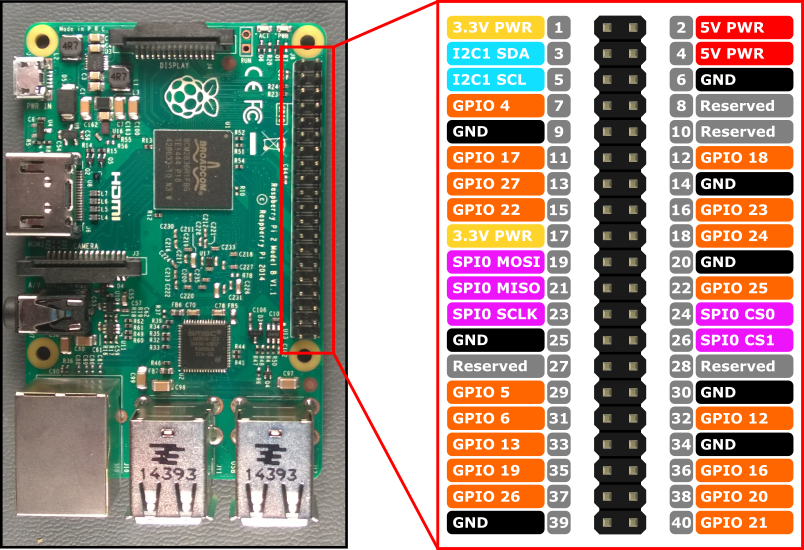
Raspberry Pi to seria jednopłytkowych komputerów wielkości karty kredytowej opracowane w Wielkiej Brytanii przez organizację Raspberry Pi Foundation z zamiarem promowania nauczania informatyki w szkołach i krajach rozwijających się. Pierwsze 2 modele komputera (Model A i Model B) zaprezentowano 29 lutego 2012 roku. Aktualnie na rynku dostępnych jest 6 różnych modeli. Najnowszy zaprezentowany 26 listopada 2015 roku to Raspberry Pi Zero, którego koszt zakupu to jedne 5$, a jego specyfikacją jest bardzo zbliżona do Modelu A lecz jego wymiary to jedyne 65 mm x 30 mm x 5 mm.

Skupmy się jednak nad najwydajniejszym modelem z całej gamy komputerów Raspberry Pi, ponieważ to on został wykorzystany w moim projekcie. Model Raspberry Pi 2 został zaprezentowany 2 lutego 2015 roku na oficjalnym blogu fundacji. Cena ustalona przez fundację za ten model to 35$. W zamian otrzymujemy komputer z czterordzeniowym, 32-bitowym procesorem ARM Cortex-A7 taktowany zegarem o częstotliwości 900Mhz opartym na architekturze ARMv7-A. Jest on ok. 4-6 razy wydajniejszy od swoich poprzedników. Dokładna specyfikacja komputera w poniższej tabeli.

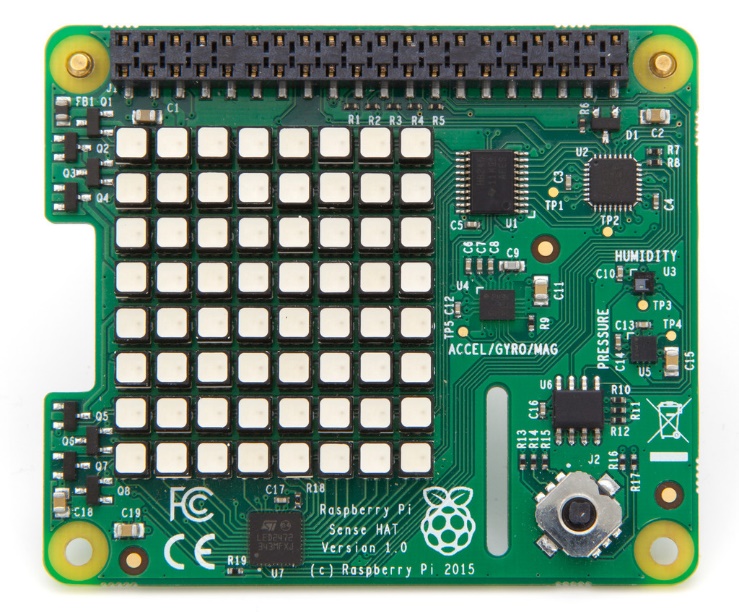
|  |  |
| --- | --- |
| SPECYFIKACJA RASPBERRY PI 2 MODEL B | |
| SoC: | Broadcom BCM2836 (CPU + GPU + DSP + SDRAM + jeden port USB) |
| CPU: | 900 MHz quad-core ARM Cortex-A7 |
| GPU: | Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decode |
| Pamięć (SDRAM): | 1 GB (współdzielona z GPU) |
| Porty USB 2.0: | 4 (uzyskane za pomocą zintegrowanego koncentratora USB) |
| Wyjścia wideo: | Composite RCA (PAL i NTSC) przez 4-pinowe (TRRS) złącze Jack, HDMI (wersja: 1.3 i 1.4) |
| Wyjścia dźwięku: | 3.5 mm jack, HDMI |
| Nośnik danych: | MicroSD |
| Połączenia sieciowe: | 10/100 Ethernet (RJ45) |
| Pozostałe złącza: | 40 x GPIO |
| Zasilanie: | 900 mA (3.1 W) |
| Źródło zasilania: | 5 V przy pomocy złącza MicroUSB, ewentualnie za pomocą złącza GPIO |
| Wymiary: | 85,60 × 53,98 mm |
| Waga: | 45 g |
| Obsługiwane systemy operacyjne: | Windows 10, Debian GNU/Linux, Fedora, Arch Linux, FreeBSD |

Raspberry Pi wykorzystuje przede wszystkim systemy operacyjne bazujące na jądrze Linuxa. Najpopularniejszy z nich, czyli Raspbian jest najbardziej rekomendowanym przez fundację systemem. Jednak w mojej pracy postanowiłem wykorzystać zupełną nowinkę, jeśli chodzi o system operacyjne na Raspberry Pi. Do świata Internet of Things postanowił w dużym stopniu włączyć się Microsoft, kiedy to w czasie premiery Raspberry Pi 2 zapowiedział swoją wersję systemu Windows 10 dla tego urządzenia. Nie były to słowa rzucone na wiatr, bowiem w kwietniu 2015 roku na konferencji Build zaprezentował pierwszą wersję systemu. Pokazuje to jak ważnym obszarem rozwoju dla Microsoftu jest internet rzeczy.

Rozmieszczenie pinów na płytce Raspberry Pi 2



Do naszej dyspozycji mamy:

* 17x – pinów GPIO
* 1x – interfejs SPI
* 1x – interfejs I2C
* 2x – zasilanie 5V
* 2x – zasilanie 3.3V
* 8x – uziemienie
  + - 1. Sense Hat
         1. 

Sense Hat to płytka rozszerzeń do Raspberry Pi. Została ona zaprojektowana specjalnie na potrzeby projektu Astro Pi. Projekt ten miał na celu edukację dzidzi, które w organizowanym przez fundację Raspberry Pi konkursie mogły wysłać swój kod w kosmos na Międzynarodową Stację Kosmiczną. Płytka ta miała umożliwić prowadzenie różnych pomiarów w kosmosie. Koszt płytki ustalony prze fundację to 30$. W skład płytki wchodzą następujące rzeczy:

* Żyroskop – zakres pomiarowy: ±245/500/2000 dps
* Akcelerometr - linowy czujnik przyśpieszenia: ±2/4/8/16 g
* Magnetometr - czujnik magnetyczny: ±4/8/12/16 gauss
* Barometr: 260 – 1260 hPa dokładność ±0.1 hPa w warunkach normalnych
* Czujnik temperatury - dokładność ± 2 °C w zakresie 0-65°C)
* Czujnik wilgotności względnej (dokładność ±4.5% w zakresie 20-80%rH, dokładność do ± 0.5 °C w zakresie 15-40 °C)
* Wyświetlacz matryca RGB LED 8x8
* Miniaturowy joystick 5 przycisków
  + - * 1. Kompletne .NET API dla tej płytki możemy pobrać z NuGet (rozszerzenie w Visual Studio do zarządzania bibliotekami).

1. Oprogramowanie
   1. W niniejszym rozdziale zostało opisane wykorzystane w pracy oprogramowane. Znajduje się tu opis takich rzeczy jak system operacyjny, po krótce środowisko programisty, lecz jest ono niezbędne do wykonania pracy oraz platforma chmurowa Microsoft Azure wykorzystywana jako pośrednik dla naszych komunikatów czyli nasz główny obiekt zainteresowania.
      * 1. Windows 10 IoT Core

Windows 10 IoT Core jest wersją systemu Windows 10 na darmowej licencji, która została zoptymalizowana dla mniejszych urządzeń z wyświetlaczem lub bez. System ten na razie działa tylko na trzech dostępnych urządzeniach: Raspberry Pi 2, MinnowBoard Max oraz DragonBoard 410c i wspiera takie architektury jak ARMv7 oraz X86. Warto zaznaczyć, że do tego zestawienia nie załapało się swego czasu intensywnie wspierane przez Microsoft urządzenie Intel Galileo, na które Microsoft wcześniej wypuścił Windowsa 8.1. Głównym powodem były zbyt wysokie wymagania sprzętowe nowego systemu. Dlaczego warto wykorzystywać produkt Microsoftu w projektach? Windows 10 IoT Core wykorzystuje Universal Windows Platform(UWP) API do budowy aplikacji. Pozwala łatwo zintegrować aplikacje z naturalnym interfejsem użytkownika, dyskami w sieci czy serwisami bazującymi na chmurze. Windows 10 IoT Core obsługuje takie języki programowania jak:

* C#
* C++
* Python
* Visual Basic
* JavaScript
* Node.js

System ten przystosowany jest do pracy z usługami Microsoft Azure, dzięki czemu można monitorować nasze zasoby w chmurze w celu poprawy wydajności i zwiększenia wydajności operacyjnej naszego systemu, a także skorzystać z innowacyjnych systemów analizy danych.

Połączenie z urządzeniem odbywa się przez przeglądarkę internetową poprzez adres IP Raspberry i port 8080. Windows Device Portal zapewnia nam podstawowe funkcje konfiguracji i zarządzania urządzeniem, dodatkowo dostęp do zaawansowanych narzędzi diagnostycznych, aby pomóc w rozwiązywaniu problemów i kontrolować wydajność urządzenia w czasie rzeczywistym. Ponieważ Windows 10 IoT nie posiada ogólnego interfejsu użytkownika tak jak inne wersje Windowsa (jedynie ten z aktualnie uruchomionej aplikacji), dlatego portal ten bardzo pomaga w użytkowaniu tego systemu. Dzięki niemu mamy możliwość między innymi zarządzania i uzyskania szczegółowych informacji o sieci, urządzeniach, uruchomionych procesach, zainstalowanych aplikacjach, ale i wydajności procesora, pamięci, układu graficznego łącza internetowego oraz interfejsu GPIO. Do naszego urządzenia możemy również łączyć się zdalnie przy użyciu PowerShell. Jest to powłoka wiersza poleceń zaprojektowana prze Microsoft specjalnie do administrowania systemem. Aby rozpocząć sesję połączenia urządzanie z PowerShell należy utworzyć relację zaufania pomiędzy nimi.

Windows 10 IoT może pracować w dwóch trybach – headed lub headless. Różnica pomiędzy tymi dwoma trybami jest obecność lub nieobecność jakiejkolwiek formy interfejsu użytkownika. Domyślnie system Windows 10 IoT Core pracuje w trybie headed i uruchamia domyślną aplikację startową, która wyświetla informacje o systemie takie jak nazwa komputera czy adres IP. W trybie tym możliwe jest wyświetlanie interaktywnych aplikacji w standardzie UWP UI. Urządzenia, która nie potrzebują funkcji interfejsu użytkownika można ustawić w trybie headless. Zmniejsza to ilość używanych zasobów urządzenia, co może przyśpieszyć działanie naszego systemu. Aplikacje headless mogą być traktowane jako usługi. System ogranicza nas do uruchomienia tylko jednej aplikacji z interfejsem użytkownika, lecz jeśli potrzebujemy więcej programów mamy możliwość uruchomienia „Background Apps”. Są to aplikacje działające w tle, których może być dowolna ilość zarówno w trybie headed jak i headless.

Włączenie trybu headless:

setbootoption.exe headless

shutdown /r /t 0

* + - * 1. Włączenie trybu headed:

setbootoption.exe headed

shutdown /r /t 0

* + - 1. C:\Users\Michał\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Visual_Studio_2012_logo_and_wordmark.svg.pngVisual Studio

Ponieważ Windows 10 IoT korzysta z Universal Windows Platform to tworzenie naszych aplikacji musi odbywać się w Microsoft Visual Studio. Jest to zintegrowane środowisko programistyczne firmy Microsoft. Od najnowszej wersji jest ono udostępnione za darmo dla wszystkich użytkowników. Dzięki temu IDE wdrażanie i debugowanie aplikacji dla Windows 10 IoT jest bardzo proste, ponieważ mamy możliwość zdalnego debugowania naszego kodu, który wykonuje się natywnie na naszym urządzeniu.

* + - 1. C:\Users\Michał\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Microsoft_Azure_Logo.svg.pngMicrosoft Azure

Microsoft Azure to platforma chmurowa stworzona przez Microsoft do budowania, wdrażania i zarządzania aplikacjami i usługami za pośrednictwem globalnej sieci rozmieszczonych centrów danych. Platforma ta zapewnia mechanizmy pozwalające składować oraz przetwarzać dane. Dane możemy przechowywać w różnych miejscach na ziemi w zależności od lokalizacji jaka nam pasuje, aby uzyskać lepszą wydajność, ponieważ usługa Azure jest dostępna na całym świecie.

Centrum IoT

Centrum IoT to usługa w platformie Microsoft Azure, dzięki której możemy łączyć się, monitorować oraz kontrolować nawet do miliona zasobów internetu rzeczy opartych na obszernym zestawie systemów operacyjnych i protokołów i miliona zdarzeń na sekundę. Dzięki temu rozwiązaniu możemy nawiązywać niezawodną, dwukierunkową komunikację między naszymi zasobami, nawet jeśli są one dostępne tylko czasowo, oraz analizować przychodzące dane i wykonywać działania na ich podstawie. Usługa zapewnia dobre bezpieczeństwo rozwiązań internetu rzeczy dzięki wykorzystaniu uwierzytelnienia na poziomie urządzeń w celu komunikowania się z urządzaniami z odpowiednimi poświadczeniami. Polecenia i powiadomienia do urządzeń mogą być przesyłane niezawodnie do podłączonych do usługi urządzeń, oraz mamy możliwość śledzenia dostarczania komunikatów dzięki potwierdzeniom doręczeń. Za pomocą rozbudowanego zestawu narzędzi dla programistów (SDK) mamy możliwość mamy możliwość łączenia urządzeń z Centrum IoT na takich platformach jak Linux czy Windows oraz systemami operacyjnymi czasu rzeczywistego przy jednoczesnym wsparciu dla takich języków jak C#, Java i JavaScript. Dodatkowo mamy możliwość wykorzystania takich protokołów jak HTTP, AMQP czy MQTT (lecz ten ostatni nie jest on domyślnie skonfigurowany). Microsoft udostępnia bezpłatnie tą usługę z ograniczeniem do 8000 komunikatów dziennie.

Jednym z głównych zadań Azure IoT Hub jest zbieranie danych telemetrycznych z urządzeń. Z tego powodu, IoT Hub jest często porównywalny do usługi w platformie Microsoft Azure, która nazywa się Event Hub. Event hub jest usługą przetwarzania zdarzeń, która zapewnia im przedostawanie się do chmury na masową skalę, z małym opóźnieniem i wysoką niezawodnością. Jednak usługi te mają wiele różnic. Między innymi różne wzorce komunikacyjne, ponieważ IoT Hub oprócz połączenia urządzenie-do-chmury może łączyć się jako chmura-do-urządzenia, a także większa skalowalność co do ilości podłączonych urządzeń z przewagą dla IoT Hub. IoT Hub zapewnia SDK dla większej ilości platform i języków.

1. Komunikacja
   1. W niniejszym rozdziale został zawarty opis protokołów wykorzystanych do komunikacji tj. protokół HTTP i AMQP. W miejscu tym chciałbym zaznaczyć, że podczas realizacji pracy okazało się, że Microsoft nie wprowadził jeszcze obsługi protokołu AMQP dla aplikacji UWP. Dlatego też na razie do wysyłania komunikatów z urządzenia do chmury musimy korzystać protokołu HTTP, lecz nie daje on nam możliwości wysyłania komunikatów z chmury do urządzenia. Myślę jednak, że nie ujmuje to w żaden sposób pracy, ponieważ możliwość przesyłania komunikatów AMQP możemy bez problemu przetestować na aplikacjach konsolowych, a kod programu przenieść do aplikacji, gdy tylko Microsoft doda obsługę tego protokołu do aplikacja UWP. Tak więc posiadanie takiej wiedzy będzie bardzo przyszłościowe oraz do wykorzystanie w innych projektach nie koniecznie na Raspberry Pi.
      * 1. Advanced Message Queuing Protocol

Advanced Message Queuing Protocol (AMQP) to otwarty standard protokołu warstwy aplikacji do przesyłania komunikatów. AMQP cechuje się ukierunkowanymi komunikatami, kolejkowaniem, routingiem (komunikaty „point-to-point” i „publish-and-subscribe”), niezawodnością i bezpieczeństwem.

* + - * 1. Najważniejsze cechy:
* Wydajny: Instrukcje protokołu i wiadomości, które przesyła są kodowane binarnie. Zawiera zaawansowane systemy kontroli przepływu, aby zmaksymalizować wykorzystanie sieci i podłączonych komponentów. Został zaprojektowany, aby zapewnić równowagę między wydajnością, elastycznością i interoperacyjności.
* Niezawodny: Pozwala na wymianę wiadomości z gwarancją niezawodności, w myśl zasady odpal i zapomnij po wysłani wiadomości czekamy na dokładnie jedno potwierdzenie doręczenia.
* Elastyczny: Jest protokołem elastycznym, ponieważ może być wykorzystywany do różnych topologii takich jak client-to-client, client-to-broker i broker-to-broker.
* Niezależny: Protokół ten nie posiada żadnych wymogów dotyczących modelu komunikacyjnego używanego przez brokera. Oznacza to, że w prosty sposób możemy dołączyć AMQP do istniejących brokerów wiadomości.

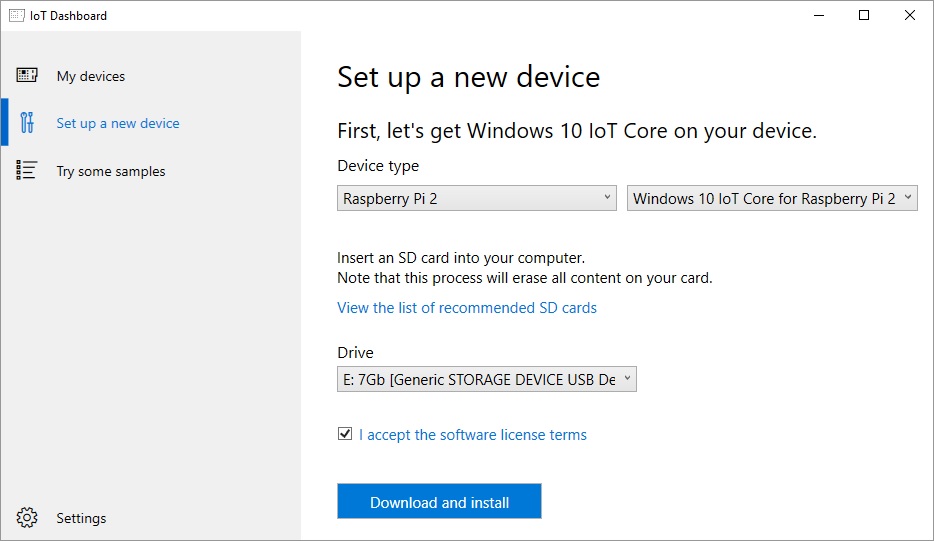
AMQP 1.0 to międzynarodowy standard, zatwierdzony przez ISO i IEC jako ISO/IEC 19464:2014. Jest on rozwijany od 2008 roku przez grupę ponad 20 firm, zarówno tych, które dostarczają technologie, ale i również takich, które tworzą rozwiązania dla użytkowników końcowych. Dzięki temu został on dostosowany do potrzeb biznesowych i współdziała z implementacjami różnych firm.

* + - 1. Hypertext Transfer Protocol

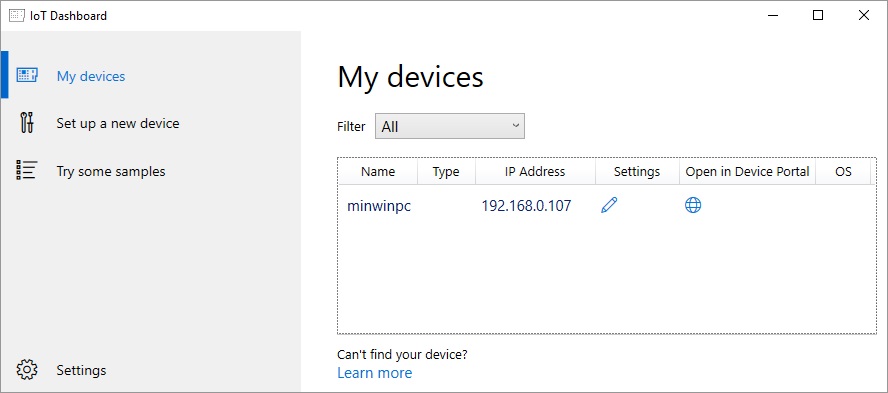
HTTP (ang. Hypertext Transfer Protocol) to bardzo dobrze wszystkim znany protokół do przesyłania dokumentów hipertekstowych wykorzystywany na co dzień jako podstawa do komunikacji danych w sieci WWW. Rozwój standardu tego protokołu był koordynowany przez Internet Engineering Task Force (IETF) i World Wide Web Consortium (W3C) i opublikowany w dokumencie RFC 2616 w 1999 roku.

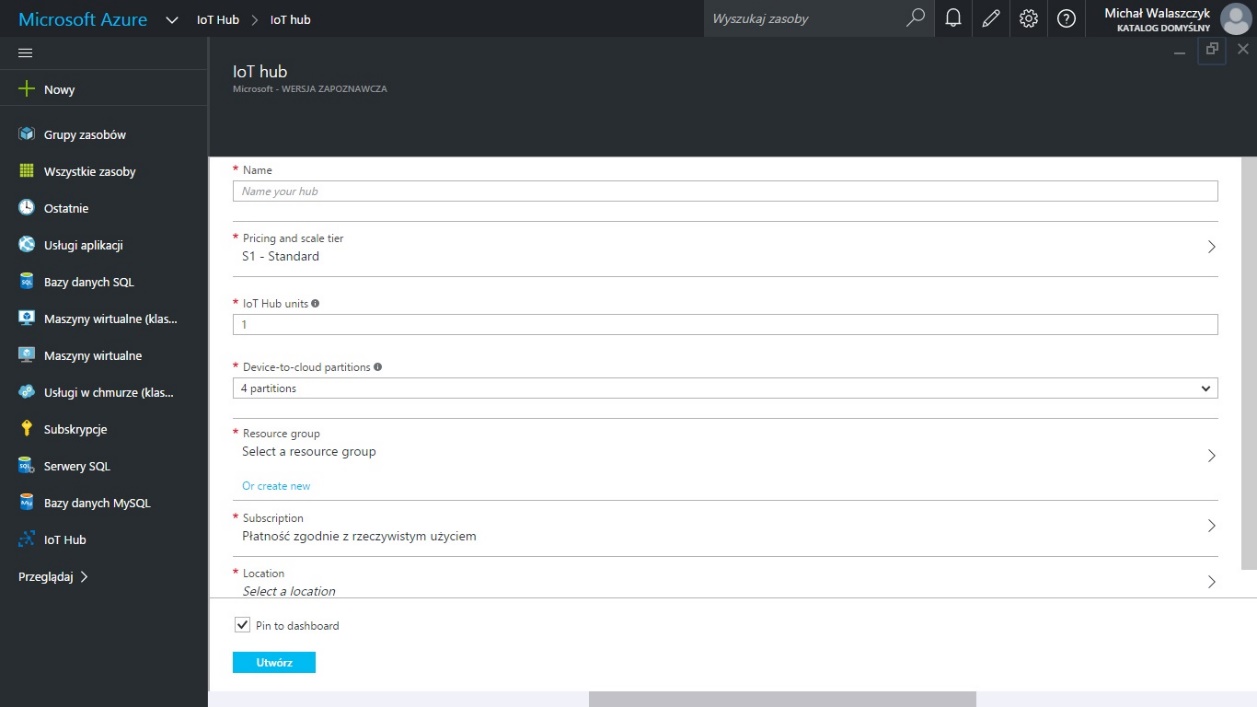
Protokół ten jest możliwy do wykorzystania w komunikacji z chmurą, ponieważ udostępnia znormalizowany sposób komunikowania się komputerów ze sobą. Jest typem protokołu bezstanowym, więc nie zachowuję żadnych informacji o poprzednich transakcjach z klientem. Do transmisji danych wykorzystuje port 80 (TCP). Microsoft nie rekomenduje używania tego protokołu lecz na razie nie mamy innej możliwości. Niestety Microsoft nie opisuje nigdzie zasady działania tego protokołu podczas komunikacji z chmurą. Podaje jedynie, że nie ma skutecznego sposobu na wdrożenie tego protokołu dla obsługi serwer push, czyli wysyłania komunikatów z chmury do urządzenia.

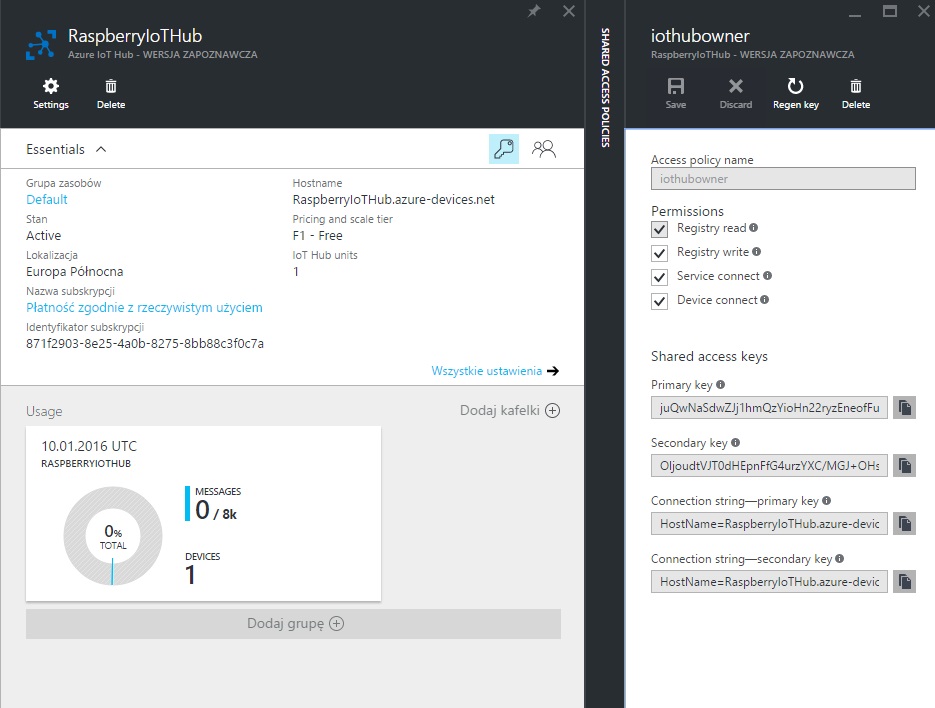
1. Realizacja
   * 1. W niniejszym rozdziale opisane są poszczególne etapy realizacji praktycznej pracy. Począwszy od instalacji najważniejszych części systemu i konfiguracji konta Azure aż do realizacji aplikacji obsługujących komunikację. Jeśli chodzi o kody programów to przedstawione są najważniejsze części, które odpowiadają za konkretne rzeczy.
        1. Instalacja Windows 10 IoT Core

Instalacja systemu na Raspberry Pi można powiedzieć, że jest dziecinnie prosta i odbywa się przez program Windows 10 IoT Core Dashboard, który sam pobiera i instaluje system na naszej karcie SD. Niestety, aby zainstalować system musimy posiadać Windowsa 10 na swoim komputerze. To jedyne płatne ograniczenie, jakie narzucił na użytkowników Microsoft.

Po skonfigurowaniu karty SD należy włożyć ją do Raspberry i uruchomić urządzenie. Po pierwszym uruchomieniu należy skonfigurować połączenie sieciowe. Jeśli chcemy korzystać z połączenia bezprzewodowego to przy pierwszym uruchomieniu należy również podłączyć ekran monitora i w ustawieniach wybrać docelową sieć WiFi. Po podłączeniu się do sieci możemy zaobserwować, że nasze urządzenie pojawiło się w programie IoT Dashboard na liście w zakładce „My devices” (jak na obrazie poniżej).

* + - * 1. 
        2. Możemy w nim zmienić w nim domyślne hasło (p@ssw0rd) do naszego urządzenia oraz jego nazwę i połączyć się z nim przez Windows Device Portal, którego możliwości opisałem już powyżej.
      1. Utworzenie IoT Hub

Aby utworzyć instancje IoT Hub w naszym koncie Microsoft Azure potrzebujemy posiadać swoje osobiste konto. Konto musi być zweryfikowane przy pomocy karty kredytowej. Potrzebujemy mieć skonfigurowaną subskrypcję konta z dla płatności zgodnie z rzeczywistym zużyciem. Dzięki temu będziemy mogli skorzystać z planu darmowego do korzystania z IoT Hub.

* + - * 1. Jedyne co musimy tutaj zrobić to podać nazwę naszej instancji, lokalizację w której będzie się ona znajdować i subskrypcję naszego konta z której chcemy skorzystać. Tworzenie IoT Hub może potrafić do paru minut. Po utworzeniu będziemy miedz następujący podgląd widoczny poniżej, w którym możemy zarządzać naszym IoT Hub. Mamy tutaj informację o ilości przesłanych wiadomości w danym dniu oraz liczbie podłączonych urządzeń. Klikając na ikonę klucza dostaniemy się do zakładki Shared access policies gdzie mamy utworzone domyślnie różne profile dla odpowiednich zasad dostępu do usługi. My możemy w ramach testów skorzystać z profilu właściciela, który ma dostęp do wszystkiego. Jest to ważna zakładka, ponieważ gdy będziemy potrzebowali Connection string, aby zarejestrować urządzenie do usługi. Możemy wybierać różne profile w zależności jakie uprawnienia chcemy ograniczyć w swojej aplikacji aby zabezpieczyć ją przed ewentualnym niepożądanym użyciem.
      1. Rejestrowanie urządzenia

Aby móc przesyłać komunikaty z urządzenia do chmury musimy najpierw zarejestrować nasze urządzenie w IoT Hub. Wykonanie tej czynności pozwoli nam na wygenerowanie unikatowego id i klucza dla naszego urządzenia. Rejestrację urządzenia robi się tylko i wyłączeni raz. Po jej wykonaniu możemy bez ograniczenia czasowego korzystać z otrzymanego klucza. Najprostszym sposobem, aby wygenerować sobie klucz i id dla swojego urządzenia to napisanie aplikacji konsolowej, którą możemy uruchomić na swoim komputerze i zapisać sobie otrzymane dane. Do realizacji takiej aplikacji będziemy potrzebowali zainstalować z NuGet menadżera biblioteki Microsoft Azure Devices. Aby je znaleźć musimy pamiętać o zaznaczeniu opcji Include prerelease, ponieważ są to biblioteki, nad którymi Microsoft prowadzi prace. Poniżej jest kod pliku Program.cs przedstawiający rejestrowanie urządzenia w IoT Hub.

using System;

using System.Threading.Tasks;

using Microsoft.Azure.Devices;

using Microsoft.Azure.Devices.Common.Exceptions;

namespace CreateDeviceIdentity

{

class Program

{

static RegistryManager registryManager;

static string connectionString = "HostName=RaspberryIoTHub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=juQwNaSdwZJj1hmQzYioHn22ryzEneofFu8oZsWGO+Y=";

static void Main(string[] args)

{

registryManager = RegistryManager.CreateFromConnectionString(connectionString);

AddDeviceAsync().Wait();

Console.ReadLine();

}

private static async Task AddDeviceAsync()

{

string deviceId = "IdUrzadzenia";

Device device;

try

{

device = await registryManager.AddDeviceAsync(new Device(deviceId));

}

catch (DeviceAlreadyExistsException)

{

device = await registryManager.GetDeviceAsync(deviceId);

}

Console.WriteLine("Wygenerowany klucz urzadzenia: {0}", device.Authentication.SymmetricKey.PrimaryKey);

}

}

}

* + - * 1. Jak widzimy, aby zarejestrować nowe urządzenie tworzymy obiekt klasy RegistryManager, który inicjalizujemy, podając w parametrze connectionString pozyskany wcześniej z portalu Azure, do metody CreateFromConnectionString. Aby móc go utworzyć potrzebujemy dodać przestrzeń nazw dodanych wcześniej bibliotek. Następnie wywołujemy asynchroniczne zadanie AddDeviceAsync().Wait(). W funkcji mamy obiekty klasy Device oraz deviceId zainicjalizowany przez nas. Aby dokonać rejestracji urządzenia do wysyłania wiadomości wywołujemy metodę registryManager.AddDeviceAsync(new Device(deviceId)). Uzyskany podczas rejestracji kod odczytujemy z obiektu device.Authentication.SymmetricKey.PrimaryKey. Otrzymany kod urządzenia oraz id musimy zapisać alby wykorzystać go w aplikacji na urządzenie. Umożliwi mu to komunikację z chmurą.
      1. Przesyłanie komunikatów z urządzenia do chmury

Mając już prawidło skonfigurowaną usługę w chmurze oraz zarejestrowane urządzenie możemy zrealizować wysyłanie komunikatów z urządzenia do chmury. Chcą robić to z przygotowanego do projektu Raspberry Pi potrzebujemy utworzyć w Visual Studio projekt aplikacji Universal App, ponieważ jak wspominałem tylko taka aplikacja ma możliwość działania na urządzeniu z systemem Windows 10 IoT Core. Po utworzeniu projektu instalujemy z menadżera NuGet biblioteki Microsoft Azure Devices Client na takiej samej zasadzie jak w poprzednim podrozdziale. Są one niezbędne do obsługi wysyłania komunikatów. Poniżej jest przedstawiona część kodu pliku SendingTelemetry.cs, który zawiera klasę wraz z funkcjami do zbierania danych z urządzeń oraz wysyłania ich do chmury. Sam opis obsługi Sense Hat opiszę w osobnym podrozdziale.

using System;

using System.Text;

using System.Threading.Tasks;

using Device.Models;

using Emmellsoft.IoT.Rpi.SenseHat;

using Microsoft.Azure.Devices.Client;

using Newtonsoft.Json;

namespace Device.Tasks

{

public class SendingTelemetry : SenseHatTask

{

private static DeviceClient deviceClient;

private static string iotHubUri = "RaspberryIoTHub.azure-devices.net";

private static string deviceKey = "Lt/i20IJ5P5kCOlP5zHJgicHTlm0eA16I3dWmip1AzQ=";

public SendingTelemetry(ISenseHat senseHat)

: base(senseHat)

{

}

public override async void Run()

{

TelemetryData telemetryData = new TelemetryData();

deviceClient = DeviceClient.Create(iotHubUri, new DeviceAuthenticationWithRegistrySymmetricKey("IdUrzadzenia", deviceKey), TransportType.Http1);

while (true)

{

//tutaj kod odpowiedzialny za pobieranie danych z sensorów

telemetryData.DeviceId = "IdUrzadzenia";

string messageString;

messageString = string.Format("Temperatura: {0}\nCisnienie: {1}\nWilgotnosc: {2}\nZyroskop: [{3},{4},{5}]\nAkcelerometr: [{6},{7},{8}]\nMagnetometr: [{9},{10},{11}]\n",

telemetryData.Temperature, telemetryData.Pressure, telemetryData.Humidity,

telemetryData.Gyro.X, telemetryData.Gyro.Y, telemetryData.Gyro.Z,

telemetryData.Acceleration.X, telemetryData.Acceleration.Y,telemetryData.Acceleration.Z,

telemetryData.Magnetic.X, telemetryData.Magnetic.Y, telemetryData.Magnetic.Z);

var message = new Message(Encoding.ASCII.GetBytes(messageString));

await deviceClient.SendEventAsync(message);

await Task.Delay(TimeSpan.FromSeconds(60));

}

}

}

}

W tym programie najważniejszą rolę pełni obiekt klasy DeviceClient, który umożliwia nam widoczność tej aplikacji jako klienta chmury. Inicjalizujemy go za pomocą metody Create(). Przyjmuje ona 3 parametry. Pierwszy to nazwa hosta naszego IoT Hub. Drugi parametr to obiekt, zainicjalizowany dwoma parametrami, klasy DeviceAuthenticationWithRegistrySymmetricKey, którego pierwszy parametr to id urządzenia, a drugi to klucz uzyskany przy rejestracji. Trzecim parametrem jest rodzaj wykorzystywanego protokołu do transportu wiadomości. W tym przypadku widnieje TransportType.Http1, ponieważ UWP nie wykorzystuje jeszcze AMQP, o czym wspominałem już wcześniej. Nasza wiadomość jest umieszczana w zmiennej messageString a następnie konwertowana na ciąg bajtów do zmiennej message. Wysłanie wiadomości wykonujemy poprzez wywołanie metody SendEventAsync() dla naszego obiektu deviceClient z przekazanym ciągiem bajtów wiadomości w parametrze funkcji. Wykonanie tej czynności powoduje wysłanie wiadomości do naszej chmury, teraz wystarczy ją już tylko odebrać na innej aplikacji. W pokazanym przykładzie kodu widzimy, że wiadomość to zwykły obiekt string, ale jeśli chcemy to mamy możliwość serializacji naszych obiektów do formatu Json i wysyłania również jako strumień bajtów.

* + - 1. Odbieranie komunikatów od urządzeń

Aby odbierać komunikaty od urządzeń również potrzebujemy connection string do naszego IoT Hub. Podobnie jak przy rejestracji urządzenia skorzystamy ze zwykłej konsolowej aplikacji dla Windowsa, aby zrealizować prostą podstawową aplikację do odbierania wiadomości od urządzeń. Ty razem szukamy i instalujemy z NuGet managera biblioteki WindowsAzure.ServiceBus. Dzięki temu uzyskujemy biblioteki pozwalające nam na obsługę usługi Service Bus, która zawiera się w naszej usłudze IoT Hub. Widać więc że do obsługi tych dwóch usług korzysta się z tych samych bibliotek. Poniżej znajduje się kod pliku Program.cs, który zawiera klasę obsługującą odbieranie wiadomości od urządzenia.

using System;

using System.Text;

using System.Threading.Tasks;

using Microsoft.ServiceBus.Messaging;

namespace ReadMessages

{

class Program

{

static string connectionString = "HostName=RaspberryIoTHub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=juQwNaSdwZJj1hmQzYioHn22ryzEneofFu8oZsWGO+Y=";

static string endpoint = "messages/events";

static EventHubClient eventHubClient;

static void Main(string[] args)

{

Console.WriteLine("Otrzymywanie widomości\n");

eventHubClient = EventHubClient.CreateFromConnectionString(connectionString, endpoint);

var partitions = eventHubClient.GetRuntimeInformation().PartitionIds;

foreach (string partition in partitions)

{

ReceiveMessages(partition);

}

Console.ReadLine();

}

private async static Task ReceiveMessages (string partition)

{

var messagesReceiver = eventHubClient.GetDefaultConsumerGroup().CreateReceiver(partition, DateTime.Now);

while (true)

{

EventData eventData = await messagesReceiver.ReceiveAsync();

if (eventData == null) continue;

string data = Encoding.UTF8.GetString(eventData.GetBytes());

Console.WriteLine(string.Format("Wiadomość. Część: {0}\n{1}", partition, data));

}

}

}

}

* + - * 1. W tym programie najważniejszą rolę pełni obiekt klasy EventHubClient. Inicjalizujemy go poprzez metodę CreateFromConnectionString() w której podajemy dwa parametry – connectionString oraz iotHubD2cEndpoint. W zmiennej partitions przechowujemy listę id partycji dla naszych wiadomości. Funkcję ReceiveMessages() musimy wywołać dla każdej partycji osobno. Wewnątrz tej funkcji tworzymy nasz odbiornik wiadomości. W metodzie CreateReceiver() podajemy partycję oraz datę, od kiedy mamy przyjmować wiadomości. Wiadomości pobieramy przy pomocy metody ReceiveAsync() i zapisujemy je do obiektu klasy EventData. Aby je przeczytać musimy pobrać z tego obiektu strumień bajtów oraz odkodować do normalnego łańcucha znaków. Teraz z przyjętą wiadomością możemy zrobić co potrzebujemy, np. wyświetlić w konsoli.

Podsumowanie

* 1. Dyskusja nad dalszym rozwojem pracy. Wnioski. Omówienie wyników. Co zrobiono w pracy i jakie uzyskano wyniki? Czy i w jakim zakresie praca stanowi nowe ujęcie problemu? Sposób wykorzystania pracy (publikacja, udostępnienie instytucjom, materiał źródłowy dla studentów). Co uważa autor za własne osiągnięcia?

Literatura

1. Przelaskowski A., Kompresja danych, skrypt internetowy http://www.ire.pw.edu.pl, 2000 – październik 2005.
2. http://www.rzepkowski.pl/elka/ - październik 2005.
3. Skarbek W., Multimedia. Algorytmy i standardy kompresji, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1998.
4. http://www.djworld.pl - październik 2005.
5. Nowak J., Kowalski A, Tytuł artykułu, Czasoppismo, 1/2004, s. 21-23.

Dodatek A. Spis zawartości dołączonej płyty CD

Praca – niniejszy dokument

* 1. www – witryny internetowe wykorzystane w pracy
  2. Progres – aplikacja systemu śledzenia ruchu w sekwencji wideo

Imię i nazwisko ………………………. Częstochowa, dn. ……………………..

Nr albumu ………………………..

Kierunek …………………………

Wydział ………………………….

Politechnika Częstochowska

**Szanowny Pan (i) Dziekan**

**OŚWIADCZENIE**

Pod rygorem odpowiedzialności karnej oświadczam, że złożona przeze mnie praca dyplomowa pt.: …............………………………………………………………………..

............................................................................................................................................

jest moim samodzielnym opracowaniem.

Jednocześnie oświadczam, że praca w całości lub we fragmentach nie została dotychczas przedłożona w żadnej szkole.

Niezależnie od art.239 Prawo o szkolnictwie wyższym wyrażam/ nie wyrażam\* zgodę na nieodpłatne wykorzystanie przez Politechnikę Częstochowską całości lub fragmentów w/w

pracy w publikacjach Politechniki Częstochowskiej.

…………………………

Podpis

\*/ nieodpowiednie skreślić.