Dodatkowe informacje. Aplikacja została opisana przez Jeremy'ego Walla pod adresem ***https://goo.gl/1aqdJ4***, a kod źródłowy wraz plikami konfiguracyjnymi można znaleźć pod adresem ***https://goo.gl/H7t5Z6***. Wideo z praktycznych testów zostało opublikowane na YouTube pod adresem ***https://goo.gl/dFaCoh***.

Kot z dostępem do Internetu

Choć prima aprilis już minął, publikujemy opis projektu, który na pierwszy rzut oka będzie wydawał się żartem. I fakt – opisywany w nim projekt powstał jako żart, ale mamy wrażenie, że sama idea jest godna zastanowienia i może być podstawą do opracowania bardziej sensownych i przydanych rozwiązań. Dlatego prezentujemy „Kota IoT", na którego pomysł wpadł Jeremy Wall z Kalifornii.

Ludzie coraz chętniej kupują nowoczesne zabawki elektroniczne, takie jak autonomiczne drony. Często tylko po to, by nagrywać z ich użyciem wideo. Jednocześnie coraz więcej rzeczy jest połączonych z Internetem, tworząc Internet of Things, czyli po polsku – Internet Rzeczy. Monitorujemy w ten sposób stan maszyn, położenie pojazdów, warunki panujące w inteligentnych budynkach itp. Monitorowanie własnego kota może przy tym wydawać się pomysłem absurdalnym, ale już proste rozwinięcie omawianej aplikacji może przynosić realne korzyści posiadaczom.

Kot z Raspberry Pi

IoT Kitteh nie jest najbardziej przemyślanym projektem, ale w naszym przekonaniu może stanowić łatwą w realizacji podstawę do dalszych eksperymentów. Całość jest bardzo łatwa w przygotowaniu i opiera się na niezmiernie popularnym minikomputerze – Raspberry Pi. Autor zastosował model RPI 2B i doposażył go w kartę sieciową Wi-Fi na USB, ale obecnie polecalibyśmy po prostu użycie nowszego Raspberry Pi 3, z wbudowaną łącznością bezprzewodową. To jednak nie wszystko. Jak wiadomo, koty chodzą własnymi drogami i trzeba się liczyć z tym, że wyjdą poza zakres domowego Wi-Fi. Dlatego IoT Kitteh dostał też modem GSM w postaci płytki Adafruit FONA 808, w której główną rolę odgrywa pokazany na **fotografii 1** moduł SimCOM SIM808. 4-zakresowy moduł pozwoli kontrolować poczynania kota, niemal niezależnie od tego, dokąd zawędruje, a zintegrowany odbiornik GPS umożliwia faktyczne śledzenie naszego podopiecznego.

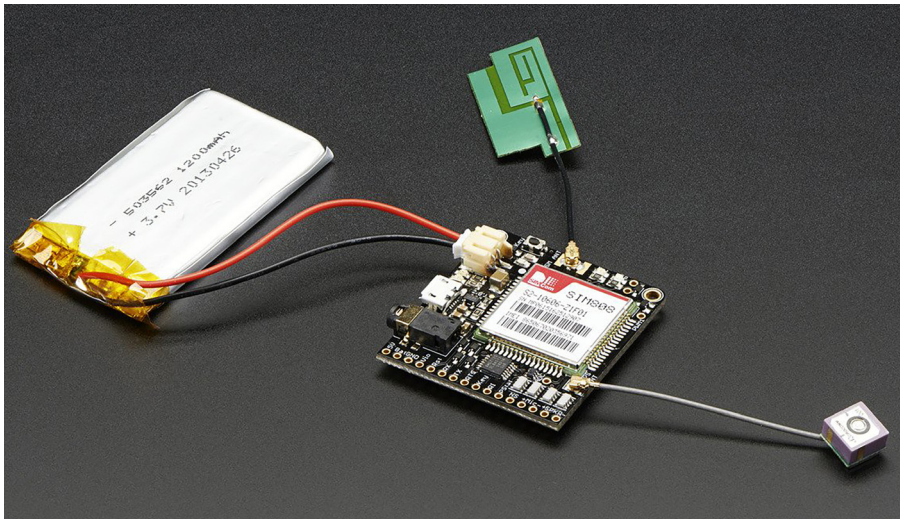
Istnieje duża szansa, że kot wejdzie do jakiejś „dziury", która ogranicza propagację fal radiowych. Stąd wypada sięgnąć po adekwatne anteny. Autor wybrał niewielką antenę o zysku energetycznym 3 dBi dla sieci GSM. Jej grubość to jedynie 2 mm. Użyta antena GPS jest kostką o wymiarach 9 mm×9 mm×6,5 mm i waży zaledwie 2,4 g.

Niemal 10 razy większe obciążenie dla kota będzie stanowić niezbędny akumulator. Tu wybór padł na ogniwo litowo-polimerowe o pojemności 1200 mAh i napięciu znamionowym 3,7 V. Przy pełnym naładowaniu napięcie to dochodzi do 4,2 V, co w praktyce przekłada się na ok. 4,5 Wh pojemności. Producent tego akumulatora zaleca ładowanie stałym prądem, nie większym niż 500 mA (ewentualnie do 0,5×C), gdyż ogniwo nie ma wbudowanego termistora. Prąd ten z łatwością można uzyskać z klasycznego złącza USB. Nie jest to jednak jedyne wymagane źródło energii. Potrzebny będzie też dowolny powerbank z gniazdem USB do zasilania samego Raspberry Pi, ponieważ wspomniany wcześniej akumulator posłuży tylko do zasilania modułu GSM/GPS.

Modem GSM w projekcie ma jeszcze dodatkowe zastosowanie – chodzi o sterowanie kotem. Jednakże jest to możliwe tylko przy założeniu, że zwierzę reaguje na komendy głosowe. Autor wykorzystuje łącze komórkowe nie tyle do danych, co do komunikacji głosowej. Dlatego niezbędnym elementem projektu jest zestaw słuchawkowy. Pozostałe komponenty to: płytki i przewody do wykonania połączeń, a w tym interfejs USB –> TTL. Potrzebna jest też karta micro SD na system operacyjny, uprząż do założenia na zwierzę i przede wszystkim sam kot (choć na psie system też powinien się sprawdzać). Jeśli chodzi o oprogramowanie, zastosowano usługi Amazon Web Services AWS IoT i bazę DynamoDB, o której także warto napisać kilka słów.

Zasada działania

Projekt IoT Kitteh ma w gruncie rzeczy bardzo ograniczony zestaw funkcji, ale wykorzystane w nim mechanizmy można bez problemu rozbudować o kolejne zastosowania – prawdopodobnie nawet znacznie bardziej przydatne. Niemniej twórca zdecydował, że zależy mu przede wszystkim na lokalizacji kota oraz przywoływaniu go.



Fotografia 1. Moduł modemu GSM Fona 808 z układem SimCom SIM808, antenami i akumulatorem

Fotografia 2. Uprząż na kota z zamontowanymi elementami projektu

Wbudowany odbiornik GPS pozwala monitorować aktualne położenie zwierzęcia. Gdy kot jest gdzieś schowany, nie trzeba też go wołać ani nawet iść w jego kierunku. Zastępuje to funkcja przywoływania głosowego, opartego właśnie na telefonii komórkowej. Tu autor natrafił na jeden problem techniczny. O ile do modemu z modułem SimCOM SIM808 można bez problemu wykonywać połączenia, to niestety kot nie umie ich samemu odbierać. W związku z tym projektant zastosował odwrotną procedurę. Połączenie głosowe jest inicjowane przez modem na kocie, który automatycznie wybiera numer telefonu swojego właściciela. Dzieje się to wtedy, gdy Raspberry Pi odbierze odpowiednie polecenie z Internetu. Inaczej mówiąc – właściciel klika w przeglądarce na przycisk połączenia, komputer na kocie wykonuje otrzymaną komendę i nawiązuje połączenie, a właściciel odbiera telefon. Dwukierunkowa komunikacja pozwala mu wydawać kotu polecenia i słyszeć ewentualne reakcje. Choć skuteczność pomysłu może wydawać się wątpliwa, doświadczenia twórcy IoT Kitteh pokazują, że metoda świetnie się sprawdza i kot faktycznie przybiega po zawołaniu go.

Montaż projektu

Połączenia pomiędzy większością komponentów są oczywiste. Na szczególną uwagę zasługuje sposób podłączenia modemu GSM. Użyty kabel USB –> TTL ma cztery żyły: RX, TX oraz dodatnie i ujemne bieguny zasilania. Należy je podłączyć odpowiednio do wyprowadzeń TX, RX VIO i GND modułu FONA808. Ponadto pin GND modułu należy połączyć z jego pinem KEY. Następnie do modułu dołączamy dwie anteny, zestaw słuchawkowy i akumulator. Wszystkie te wyprowadzenia są jednoznacznie opisane.

Tymczasem do Raspberry Pi należy dołączyć kartę Wi-Fi na USB (chyba że korzystamy z RPI 3), wtyczkę USB kabla USB –> TTL i źródło zasilania. Oczywiście potrzebna będzie też karta pamięci z wcześniej wgranym oprogramowaniem: Raspbianem Jessie i kodem programu. Całość została umieszczona w odblaskowej uprzęży dla kota, jak na **fotografii 2**, a same płytki autor ochronił za pomocą kartonowych pudełek. To mało eleganckie rozwiązanie, ale na etapie prototypowania może wystarczyć.

Zastosowane aplikacje

Dla doświadczonego elektronika omawiany projekt od strony sprzętowej jest bardzo prosty i jego realizacja nie wiąże się z praktycznie żadnymi wyzwaniami – być może z wyjątkiem problemu namówienia kota do współpracy. Znacznie ciekawiej IoT Kitteh prezentuje się od strony oprogramowania. Stanowi świetny przykład wykorzystania protokołów i narzędzi, których nigdy dotąd nie omawialiśmy na łamach EP, a które zostały zaprojektowane właśnie z myślą o IoT. Użyte techniki są nowoczesne i dawniej byłyby uznane za niekonwencjonalne, ale obecnie warto je poznać, by móc lepiej przygotowywać aplikacje Internetu Rzeczy.

Przede wszystkim autor napisał kod programu w JavaScripcie. Nie jest długi – źródło zajmuje jedynie 157 linijek! Sekret polega na wykorzystaniu gotowych bibliotek oraz narzędzi do zarządzania nimi. Kod uruchamiany jest na serwerze Node.js, włączonym na Raspberry Pi, na kocie. Program komunikuje się z chmurą firmy Amazon – AWS IoT, dokąd przekazuje i skąd pobiera dane. AWS IoT jest w stanie zapewnić dwukierunkową, bezpieczną komunikację w oparciu na „lekkach" protokołach. Zbierane dane są natomiast gromadzone w tzw. nieSQL-owej bazie Amazon DynamoDB. Wymiana danych odbywa się z użyciem protokołu MQTT, który bazuje na formacie json i jest wspierany przez AWS IoT. Za zapewnienie bezpieczeństwa odpowiadają certyfikaty zgodne ze standardem x509, a więc oparte na kluczu prywatnym i publicznym. Korzysta z nich zarówno aplikacja w JavaScripcie, działająca na Node.js na RPI, jak i terminal wykorzystywany do przesyłania poleceń do kota. Autor w tym celu użył narzędzia MQTT.fx – napisanego w Javie klienta protokołu MQTT (**rysunek 3**).

Zaletą użytych narzędzi jest fakt, że da się ich wszystkich użyć bezpłatnie. Dostęp do serwera AWS IoT jest obecnie bezpłatny przez pierwsze 12 miesięcy użytkowania, z tym że z miesięcznym limitem 250 tysięcy przesyłanych komunikatów. Natomiast baza

Listing 1. Główny kod programu javascriptowego

```
// ładowanie bibliotek
var iot = require(,aws-iot-device-sdk');
var _ = require(,lodash');
var SimCom = require(,simcom').SimCom;

// konfiguracja połączenia z serwerami AWS IoT
var device = iot.device({
    keyPath: ,/home/pi/AWSCerts/privateKey.pem',
    certPath: ,/home/pi/AWSCerts/cert.pem',
    caPath: ,/home/pi/AWSCerts/root.pem',
    clientId: ,rpi01',
    region: ,us-east-1'
});

// połączenie z modułem GSM poprzez kabel USB
var simcom = new SimCom(,/dev/ttyUSB0');

// przypisanie akcji logowania informacji do zdarzenia otwarcia połączenia z modułem GSM
simcom.on(,open', function() {
    console.log(,simcom open');
});

// przypisanie akcji logowania informacji do zdarzenia błędu połączenia w module GSM
simcom.on(,error', function() {
          console.log(,simcom error');
});

// funkcja wywołująca komendę AT
function executeAtCmd(atCmd) {
    // logowanie informacji o uruchamianej komendzie
    console.log(,Execute AT Command: [, + atCmd + ,]');
    // wysłanie komendy AT do modemu GSM
    simcom.modem.execute(atCmd).then(function(lines) {
        // w przypadku powodzenia:
        // logowanie informacji zwrotnej z modemu
        console.log(,AT Response', lines);
        // przesłanie odpowiedzi modemu do serwera w chmurze
        sendToServer(,AT Response: , + JSON.stringify(lines));
    }, function(error) {
        // w przypadku błędu:
        // logowanie informacji o błędzie z modemu
        console.error(,AT Command Error', error);
        // przesłanie zwróconego przez modem błędu do serwera w chmurze
        sendToServer(,AT Command Error: , + JSON.stringify(error));
    });
}

// funkcja wysyłająca wiadomość SMS przez modem
function sendSms(to, message) {
    console.log(,Sending SMS: , + to + , : , + message);
    simcom.sendSMS(to, message);
    sendToServer(,Sending SMS: , + to + , : , + message);
}

// przesłanie wiadomości do serwera w chmurze, do tematu subskrybowanego przez administratora
function sendToServer(message) {
    device.publish(,test/topic1', message);
}

// przypisanie akcji do zdarzenia nawiązania połączenia z serwerem AWS IoT
device.on(,connect', function() {
    console.log(,iot: connect');
    // rozpoczęcie subskrypcji tematu używanego do przekazywania wiadomości do kota
    device.subscribe(,test/topic2');
    // przesłanie wiadomości do chmury, informującej administratora o rozpoczęciu nasłuchiwania komend przez kota
    sendToServer(,iot: connect');
});

// przypisanie akcji do zdarzenia nadejścia wiadomości z chmury
device.on(,message', function(topic, payload) {
    // logowanie treści otrzymanej wiadomości
    console.log(,iot: message', topic, payload.toString());
    // przesłanie do chmury informacji powiadamiającej administratora o otrzymanej wiadomości
    sendToServer(,Received: , + payload.toString());
    // zdekodowanie treści otrzymaje wiadomości (wykorzystywany jest format JSON)
    var req = JSON.parse(payload.toString());
    // pobranie treści atrybutu określającego typ nadchodzącej wiadomości
    switch (_.get(req, ,type', null)) {
        // w przypadku gdy wiadomość dotyczy funkcji telefonu:
        case ,phone': {
            // pobranie treści polecenia związanego z telefonem
            switch (_.get(req, ,cmd', null)) {
                // operacje w przypadku, gdy otrzymano polecenie wybrania numeru
                case ,dial': {
                    // poobranie numeru telefonu do wybrania
                    var to = _.get(req, ,to', null);
                    if (!_.isNull(to)) {
                        // polecenie modemowi GSM (za pomocą komendy AT) nawiązania połączenia telefonicznego z numerem
                        executeAtCmd(,ATD' + to + ,;');
                    }
                    break;
                }
                // operacje wykonywane, gdy otrzymano polecenie rozłączenia rozmowy
                case ,hangup': {
                    // polecenie modemowi GSM, aby rozłączył rozmowę
                    executeAtCmd(,ATH');
                    break;
                }
                // operacje wykonywane, gdy otrzymano polecenie sprawdzenia jakości sygnału komórkowego
                case ,signal': {
                    // wysłanie modemowi GSM komendy AT, która zwraca informację o poziomie sygnału
                    executeAtCmd(,AT+CSQ');
                    break;
                }
                // operacje wykonywane, gdy otrzymano polecenie sprawdzenia poziomu naładowania akumulatora
                case ,battery': {
                    // wysłanie modemowi GSM komendy AT, która zwraca informację o poziomie naładowania akumulatora
                    executeAtCmd(,AT+CBC');
                    break;
                }
                // operacje wykonywane, gdy otrzymano polecenie wysłania wiadomości SMS
```

Listing 1. cd.

```
                case ‚sms’: {
                    // pobranie numeru telefonu, pod który ma być wysłana wiadomość SMS
                    var to = _.get(req, ‚to’, null);
                    // pobranie treści wiadomości SMS do wysłania
                    var message = _.get(req, ‚message’, null);
                    if (!_.isNull(to) && !_.isNull(message)) {
                        // wywołanie funkcji wysyłającej SMS o zadanej treści, pod zadany numer
                        sendSms(to, message)
                    }
                    break;
                }
            }
            break;
        }
        // w przypadku, gdy wiadomość dotyczy funkcji związanej z GPS-em
        case ‚gps’: {
            // sprawdzenie treści otrzymanej komendy dla GPS-u
            switch(_.get(req, ‚cmd’, null)) {
                // operacje w przypadku otrzymania komendy włączającej odbiornik GPS
                case ‚on’: {
                    // wysłanie do modemu komendy AT włączającej GPS
                    executeAtCmd(‚AT+CGNSPWR=1’);
                    break;
                }
                // operacje w przypadku otrzymania komendy wyłączenia odbiornika GPS
                case ‚off’: {
                    // wysłanie do modemu komendy AT wyłączającej GPS
                    executeAtCmd(‚AT+CGNSPWR=0’);
                    break;
                }
                // operacje w przypadku otrzymania komendy sprawdzenia, czy odbiornik GPS jest włączony
                case ‚power’: {
                    // wysłanie do modemu komendy AT pobierającej stan (włączenia/wyłączenia) odbiornika GPS
                    executeAtCmd(‚AT+CGNSPWR?’);
                    break;
                }
                // operacje w przypadku otrzymania komendy sprawdzenia informacji stanu z GPS-u
                case ‚info’: {
                    // wysłanie do modemu komendy AT pobierającej koordynaty i inne parametry z odbiornika GPS
                    executeAtCmd(‚AT+CGNSINF’);
                    break;
                }
            }
            break;
        }
    }
});
```

DynamoDB jest bezterminowo udostępniana bezpłatnie, o ile tylko ilość przechowywanych w niej danych nie przekracza 25 GB. Naturalnie darmowe są też serwer Node.js oraz program MQTT.FX.

Struktura programu

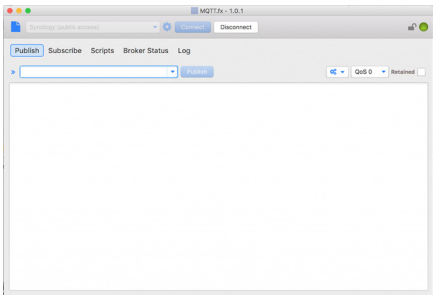
Kod programu javascriptowego, uruchomionego na Node.js, a więc i główny element całej aplikacji został zaprezentowany na **listingu 1**. Auto skorzystał z dwóch głównych bibliotek i jednej pomocniczej:

AWS-IOT-DEVICE-SDK to biblioteka przygotowana przez Amazona, przystosowana do obsługi chmury AWS IoT za pomocą kilku komend. Jest lekka i w razie potrzeby można znaleźć jej odpowiedniki dla innych języków programowania.

SIMCOM to biblioteka Javascriptowa przygotowana do obsługi modułu GSM firmy SimCom.

LOADASH to biblioteka Javascriptowa ułatwiająca posługiwanie się JavaScriptem – autor używa jej praktycznie tylko do uproszczenia funkcji pobierania atrybutów z obiektów i nie ma ona realnego znaczenia dla implementacji projektu.

Program sprowadza się do konfiguracji chmury AWS IoT i połączenia z modemem GSM, po czym następuje przypisanie akcji do zdarzeń, takich jak nadejście wiadomości, błąd modemu GSM czy otrzymanie polecenia z chmury. Działanie systemu jest właśnie oparte na zdarzeniach. Dodatkowo autor przyjął zasadę, że właściwie wszystko, co dzieje się po stronie kota,



Rysunek 3. Główne okienko programu MQTT.FX

jest przekazywane do chmury w postaci prostych komunikatów, które otrzymuje dowolny subskrybujący je na serwerze administrator. W ten sposób polecenia kotu można przesyłać z więcej niż jednego urządzenia jednocześnie i na wielu urządzeniach naraz monitorować ruch zwierzaka.

W momencie nadejścia komunikatu z chmury program kota analizuje go i rozpoznaje komendę. Dostępne polecenia to:

- Nawiązanie połączenia ze wskazanym numerem.
- Rozłączenie połączenia.
- Sprawdzenie poziomu sygnału komórkowego.
- Sprawdzenie napięcia na akumulatorze.
- Wysłanie SMS na wskazany numer.
- Włączenie odbiornika GPS.
- Wyłączenie odbiornika GPS.
- Sprawdzenie, czy odbiornik GPS jest włączony.
- Pobranie danych z odbiornika GPS.

Polecenia te są tłumaczone na odpowiednie komendy AT i przesyłane od modułu SimComa, a odpowiedź jest automatycznie przekazywana do chmury – poprzez połączenie internetowe.

Podsumowanie

Projekt, choć może wydawać się żartem, jest bardzo zgrabnie napisany i naprawdę niewiele trzeba, by go odtworzyć. Co więcej, stanowi świetny wzór dla innych aplikacji tego typu, które mogą działać na identycznej zasadzie, ale w bardziej szczytnych celach. W podobny sposób można by było np. zrealizować system wspomagający ludzi starszych. Z łatwością można też przygotować jego inne wersje, oparte na innych językach programowania, a nawet przygotować analogiczny program dla urządzenia bez systemu operacyjnego. Kluczowe jest tylko zdobycie odpowiednich bibliotek dla modułu GSM/GPS oraz przygotowanie zestawu funkcji do obsługi komunikacji z chmurą. Ta natomiast – ze względu na wykorzystanie MQTT – jest bardzo prosta, więc lekka rozbudowa powszechnie dostępnych bibliotek do komunikacji HTTP przez sieć powinna wystarczyć. Drobnym problemem może być wymuszenie szyfrowania i autoryzacji, ale i z tym można sobie poradzić.

Biorąc pod uwagę wielkość programu i elementy potrzebne do jego realizacji, można zdecydowanie polecić go jako przykład (trening) dla wszystkich czytelników, którzy chcieliby zacząć tworzyć systemy Internetu Rzeczy.

Marcin Karbowniczek, EP