POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko studenta: Jan Nielek
Nr albumu: 139643
Studia pierwszego stopnia
Forma studiów: niestacjonarne
Kierunek studiów: Informatyka
Specjalność/profil: -

PROJEKT DYPLOMOWY INŻYNIERSKI

Tytuł projektu w języku polskim: System monitorowania domu z wykorzystaniem układów ESP8266 i platformy dla systemów wbudowanych Raspberry Pi

Tytuł projektu w języku angielskim: Home monitoring system using ESP8266 and Raspberry Pi platform

Potwierdzenie przyjęcia projektu	
Opiekun projektu *podpis*	Kierownik Katedry/Zakładu (pozostawić właściwe) *podpis*
dr inż. Krzysztof Bikonis	

Data oddania projektu do dziekanatu:

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

OŚWIADCZENIE

Imię i nazwisko: Jan Nielek
Data i miejsce urodzenia: 30.07.1992, Człuchów
Nr albumu: 139643
Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki
Kierunek: informatyka
Poziom studiów: pierwszy
Forma studiów: niestacjonarne

Ja, niżej podpisany(a), wyrażam zgodę/nie wyrażam zgody* na korzystanie z mojego projektu dyplomowego zatytułowanego: System monitorowania domu z wykorzystaniem układów ESP8266 i platformy dla systemów wbudowanych Raspberry Pi
do celów naukowych lub dydaktycznych.[1]

Gdańsk, dnia ....
podpis studenta

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2016 r., poz. 666 z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym (Dz. U. z 2012 r., poz. 572 z późn. zm.),[2] a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładany projekt dyplomowy został opracowany przeze mnie samodzielnie.

Niniejszy projekt dyplomowy nie był wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. projekcie dyplomowym, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji projektu dyplomowego z załączoną wersją elektroniczną.

Gdańsk, dnia ....
podpis studenta

Upoważniam Politechnikę Gdańską do umieszczenia ww. projektu dyplomowego w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej oraz poddawania jego procesom weryfikacji i ochrony przed przywłaszczaniem jego autorstwa.

Gdańsk, dnia ....
podpis studenta

*) niepotrzebne skreślić

[1] Zarządzenie Rektora Politechniki Gdańskiej nr 34/2009 z 9 listopada 2009 r., załącznik nr 8 do instrukcji archiwalnej PG.
[2] Ustawa z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym:
Art. 214 ustęp 4. W razie podejrzenia popełnienia przez studenta czynu podlegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.
Art. 214 ustęp 6. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 4, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o popełnieniu przestępstwa.

Streszczenie

Internet Reczy jest coraz powszechniejszy w dzisiejszym świecie. W ramach projektu inżynierskiego został zaprojektowany system monitorujący stan domu przy pomocy sieci czujników. Podjęto próbę stworzenia energooszczędnego czujnika i zbudowania sieci sensorów od podstaw. Napisano aplikację serwerową zajmującą się obsługą czujników i użytkowników. Podjęto także rozważania na temat aktualnego stanu jak i przyszłości aplikacji.

Słowa kluczowe: Internet Rzeczy, systemy wbudowane, bazy danych, strony internetowe

Dziedziny nauki i techniki zgodnie z wymogami OECD: Elektronika, Sprzęt komputerowy i architektura komputerów

Abstract

Internet of Things is more and more prevailing in the current world. As part of the project, a home monitoring system utilizing a network of sensors was built. An attempt was made to create an energy efficient sensor and to build a sensor network from scratch. A server application that handles sensor and user communication was made. Considerations were made about current state of the application, and of the future of the project.

Keywords: Internet of Things, embedded systems, databases, web pages

Spis treści

Wykaz ważniejszych skrótów i oznaczeń

IoT – ang. *Internet of Things* Internet Rzeczy;

WiFi – ang. *Wireless Fidelity* sieć bezprzewodowa

Framework – szkielet do budowy aplikacji

Bootloader – program rozruchowy, służący do załadowania właściwej aplikacji do pamięci komputera

1. Wstęp i cel pracy

Internet Rzeczy jest jedną z najszybciej rozwijających się gałęzi sektora systemów wbudowanych. Wydatną jego częścią są tzw. Inteligentne Domy. Pod tym pojęciem mieści się szeroka gama urządzeń monitorujących wiele parametrów takich jak temperatura, wilgotność czy też nawet nasłonecznienie. Korzystając z tych danych, system może odpowiednio sterować ogrzewaniem tak, aby poprawić komfort mieszkańców przy równoczesnym zmniejszeniu zużycia energii.

Celem pracy było zaprojektowanie i wykonanie sieci sensorów temperatury i wilgotności stale badających warunki panujące w domu.

W poniższym projekcie skupiono się na monitorowaniu parametrów przy użyciu zasilanych bateryjnie mikrokontrolerów. Zaprojektowany został przykładowy moduł współpracujący z serwerem. Komunikacja odbywała się z wykorzystaniem sieci Wi-Fi. Pomiary były zapisywane do bazy danych w celu późniejszego zaprezentowania ich użytkownikowi poprzez stronę WWW.

2. Analiza projektowa w odniesieniu do stanu wiedzy

Systemy automatyki domowej są coraz popularniejsze. Są wykorzystywane do zwiększenia komfortu i poprawy bezpieczeństwa użytkowników. Inteligentny Dom jest w stanie ułatwić życie ludziom starszym lub z niepełnosprawnością i zmniejszyć nakład pracy mieszkańców poprzez zautomatyzowanie pewnych zadań. System może być zaprojektowany z wykorzystaniem centralnego kontrolera, który steruje i monitoruje wiele niezależnych urządzeń takich jak gniazda zasilania, oświetlenie, czujniki temperatury i wilgotności, czujniki dymu, gazu i ognia czy też urządzeń alarmowych. Jedną z największych zalet automatyki domowej jest to, że można nią łatwo sterować za pomocą całej gamy urządzeń poczynając od telefonów, tabletów czy też komputerów PC. Szybki rozwój technologii bezprzewodowych pozwala na zdalne zarządzanie systemem. W podrozdziałach przedstawiono selekcję najbardziej odpowiednich technologii i rozwiązań wykorzystanych w projekcie. Selekcji dokonano na podstawie aktualnego stanu wiedzy.

2.1. Medium transmisyjne sensor – serwer

Podstawowym problemem, z którym zmierzyć musi się projektant sieci sensorów jest sposób komunikacji pomiędzy składowymi systemu. Podziału można dokonać na komunikację przewodową i bezprzewodową. Ze względu na mobilność i łatwość w realizacji w projekcie wykorzystano wariant bezprzewodowy. Na rynku dominują trzy standardy: Bluetooth, WiFi (ang. *WirelessFidelity*) i ZigBee. Selekcję najodpowiedniejszego z nich do projektu wykonano wg trzech kryteriów: poboru mocy, powszechności użycia i kosztów implementacji.

2.1.1.Kryterium poboru mocy

Wprowadzony na rynek standard Bluetooth Low Energy pozwala na zastosowanie urządzeń zasilanych z baterii pastylkowych, które działają nawet ponad 10 lat [1]. ZigBee oferuje także niskie zużycie energii, na poziomie 2 lat zasilania z wykorzystaniem dwóch baterii AA. WiFi nie jest energooszczędne, jednak przy zastosowaniu odpowiednich trybów uśpienia i oszczędzania energii można osiągnąć półroczny czas pracy bez ładowania, co zostało dowiedzione eksperymentalnie i opisane w podrozdziale 5.3

2.1.2.Kryterium powszechności użycia

Obecnie, każde gospodarstwo domowe podłączone do sieci Internet posiada punkt dostępu do sieci WiFi. Dzięki powszechności tego rozwiązania, dodanie nowych urządzeń do systemu jest stosunkowo łatwe. Bluetooth i ZigBee wymagają utworzenia nowej sieci i podłączenia do niej serwera, a także tunelowania do sieci IP z racji wykorzystania innych protokołów niż w WiFi.

2.1.3.Kryterium kosztów implementacji

Obecnie na rynku dostępne są zarówno tanie mikrokontrolery wspierające sieć WiFi (np. ESP8266 firmy Espressif), jak i mikrokontrolery wspierające sieć Bluetooth (nRF52 firmy Nordic

Semiconductors). Rozwiązania wspierające sieć ZigBee są droższe, co przesądziło o odrzuceniu tego standardu w poniższym projekcie. Na rzecz WiFi przemawia możliwość wykorzystania istniejącej infrastruktury w gospodarstwie domowym.

W odniesieniu do powyższych podpunktów, zadecydowano o wykorzystaniu WiFi jako medium transmisyjnego pomiędzy sensorami a serwerem.

2.2. Wybór mikrokontrolera

Obecnie większość z producentów ma w ofercie mikrokontroler wspierający sieć WiFi. Są to np. Particle Photon (rdzeń ARM Cortex-M3), WiFiMCU (Cortex-M4), czy też układ firmy Espressif ESP8266 (Tensilica L106) [2]. Spośród wymienionych, ten ostatni charakteryzuje się najniższą ceną. Do innych jego zalet należy wsparcie dla biblioteki Arduino [3], stosunkowo niewielka ilość zewnętrznych elementów wymaganych do uruchomienia [4], czy też wsparcie dla trybu głębokiego uśpienia (*Deep Sleep,* [2]). Z tych powodów, został on wybrany jako główny układ sterujący dla sensorów.

2.3. Protokół transmisji danych pomiędzy sensorami a serwerem

Ze względu na wybór sieci WiFi jako medium transmisyjnego, komunikacja pomiędzy składowymi systemu może odbywać się w oparciu o warstwę sieciową modelu OSI (warstwy internetowej w modelu TCP/IP). W takim przypadku można wykorzystać unikalny adres MAC każdego z sensorów do jego identyfikacji. Obecnie w przemyśle nie istnieje jeden standard dla komunikacji, wielu producentów korzysta z opracowanych już protokołów takich jak np. MQTT (ang. *Message Queuing Telemetry Transport*, transport telemetrii w oparciu o kolejkowanie wiadomości). Organizacje takie jak Open Mobile Alliance próbują wdrożyć własne standardy, a jeszcze inni producenci wdrażają własnościowe, zamknięte protokoły [5].

2.3.1. Własny protokół w oparciu o TCP/IP

Pomimo wielu korzyści własnej implementacji takich jak pełna kontrola nad ilością danych, ścisłe dopasowanie do sprzętu czy też możliwe najniższe zużycie energii, takie rozwiązanie posiada wiele wad. Wśród nich wyróżnić można: czas pracy potrzebny na prawidłowe zaprojektowanie i przetestowanie takiego protokołu, czy też potencjalne problemy w przyszłości przy zmianie architektury systemu. Z tego względu to rozwiązanie zostało odrzucone na etapie planowania projektu.

2.3.2. Interfejs REST

Model REST (ang. *Representational State Transfer*, zmiana stanu poprzez reprezentację) to sposób na zapewnienie bezstanowej komunikacji za pomocą zorganizowanego i jawnie określonego zbioru operacji [6]. Znajduje on wykorzystanie głównie w aplikacjach webowych. Ze względu na to, że komunikacja odbywa się przy pomocy protokołu HTTP (ang. *Hypertext Transfer Protocol*, protokół transmisji hipertekstu) świetnie sprawdza się przy interakcji na stronach WWW. Niestety, wykorzystanie HTTP wiąże się ze zwiększeniem ilości wysyłanych

danych ze względu na dodatkowe nagłówki [7]. Z tego powodu, nie jest on idealny dla systemów wbudowanych.

2.3.3.MQTT

MQTT to standard bardzo prostego protokołu transmisji danych [8]. Został on zaprojektowany specjalnie dla sieci M2M (ang. *Machine to machine*, interfejs maszyna-maszyna), czy też IoT (ang. *Internet of Things*, Internet Rzeczy). Charakteryzuje się niskim zapotrzebowaniem na energię, prostotą tworzenia i odbierania wiadomości. Na jego niekorzyść świadczy to, że wymaga do poprawnego działania zewnętrznego brokera wiadomości.

Ze względu na istniejące już biblioteki wspierające protokół MQTT na mikrokontroler ESP8266,
a także łatwość implementacji jego obsługi po stronie serwera, został on wykorzystany w projekcie.

2.4. Architektura aplikacji serwerowej

W założeniach projektowych, serwer powinien obsługiwać trzy funkcjonalności, mianowicie: odbiór danych z sensorów, zapis i odczyt z bazy danych, a także prezentację odczytów użytkownikowi. W celu możliwie największej przenośności na inne architektury sprzętowe założono wykorzystanie języka Python w wersji 3.6. Umożliwia to zarówno uruchomienie aplikacji serwerowej na systemach Windows, jak i Linux z procesorami z rodzin ARM i x86. Python oferuje wiele platform programistycznych do wdrażania aplikacji webowych, jedną z nich jest Flask [9]. Flask jest określany przez twórców jako mikroframework i nie wymaga dużych zasobów sprzętowych jak i nakładu pracy, aby uruchomić na nim prostą stronę WWW. Na jego korzyść świadczy także duża ilość dodatkowych bibliotek rozszerzających funkcjonalność. Temat konfiguracji, uruchomienia i zewnętrznych bibliotek dla serwera został szerzej opisany w rozdziale trzecim.

Z racji wykorzystania w projekcie protokołu MQTT, potrzebny był broker wiadomości, czyli serwer odpowiedzialny za dystrybucję wiadomości. W sieci dostępnych jest wiele darmowych brokerów [10], jednakże z powodu ograniczenia się do jednej sieci LAN (ang. *Local Area Network*, lokalna sieć komputerowa) uruchomiono własnego brokera: mosquitto [11], równolegle do działającego serwera głównego aplikacji. Pozwoliło to na zwiększenie bezpieczeństwa jak i kontroli nad całą siecią sensorów, a także ułatwiło diagnostykę poprzez dostęp do pełnych logów serwera.

Jako serwer bazy danych wykorzystano popularny silnik MySQL [12].

2.5. Platforma sprzętowa serwera

Zgodnie z założeniami projektowymi jako serwer służyć może dowolna współczesna architektura sprzętowa, która wspiera systemy Windows lub Linux. Z tego względu jako platforma deweloperska wykorzystany został system Windows (procesor x86), co pozwoliło na szybkie prototypowanie i testowanie kodu lokalnie. Jako serwer produkcyjny wykorzystany został mikrokomputer Raspberry Pi (procesor ARM) z systemem Raspian (dystrybucja Linuxa oparta o Debiana). Pozwoliło to znacząco zmniejszyć koszty utrzymania całego systemu ze względu wielokrotnie mniejsze zużycie energii względem komputera PC.

3. Implementacja

Projekt można podzielić na dwie części – sensory i serwer. Całość została zaprojektowana na przełomie czerwca i lipca 2017.

3.1. Sensor temperatury i wilgotności

Jako przykładowy układ, zaprojektowany został zasilany bateryjnie sensor temperatury i wilgotności wykorzystujący układ Si7021 [13], komunikujący się z mikrokontrolerem ESP8266 za pomocą interfejsu I^2C. Sensor został także wyposażony w układ ładowania ogniw litowo-polimerowych lub litowo-jonowych [14]. Ładowanie odbywało się z wykorzystaniem złącza microUSB. Program kontrolujący pracę mikrokontrolera został napisany w języku C++, z wykorzystaniem biblioteki Arduino.

3.1.1. Schemat blokowy urządzenia

Pierwsze prototypy sensora powstały z wykorzystaniem płytki stykowej. Jednakże nie było to rozwiązanie praktyczne, szczególnie przy długotrwałych pomiarach. W celu uzyskania powtarzalnych wyników został zaprojektowany schemat blokowy, który został potem wykorzystany do stworzenia płytki drukowanej. Schemat został umieszczony jako Dodatek A.

W celu łatwego prototypowania, wyprowadzone zostały następujące interfejsy: UART, I^2C i SPI. Ponadto, aby ułatwić programowanie, wyprowadzony został IO0 jako specjalny pin. Za jego pomocą bootloader decydował o tym, w jakim trybie ma zostać uruchomiony ESP8266 po restarcie. W przypadku, kiedy IO0 będzie w stanie niskim (zero logiczne), bootloader będzie oczekiwał na aktualizację oprogramowania przez połączenie szeregowe, w przeciwnym przypadku uruchomi się normalnie. Na szczególną uwagę zasługuje także podłączenie pinu IO16 do RST. ESP8266 oferuje tryb tak zwanego głębokiego uśpienia, w którym drastycznie ogranicza zużycie prądu. Podłączenie pinu IO16 do RST pozwala na wybudzenie mikrokontrolera z tego trybu. Jedyne wejście analogowe dostępne w ESP8266, czyli pin ADC (ang. *analog to digital converter*, przetwornik analogowo cyfrowy) pozostawiono niepodłączone. Było to działanie celowe, ze względu na możliwość pomiaru napięcia zasilania poprzez wewnętrzne podłączenie tego pinu do zasilania układu [2].

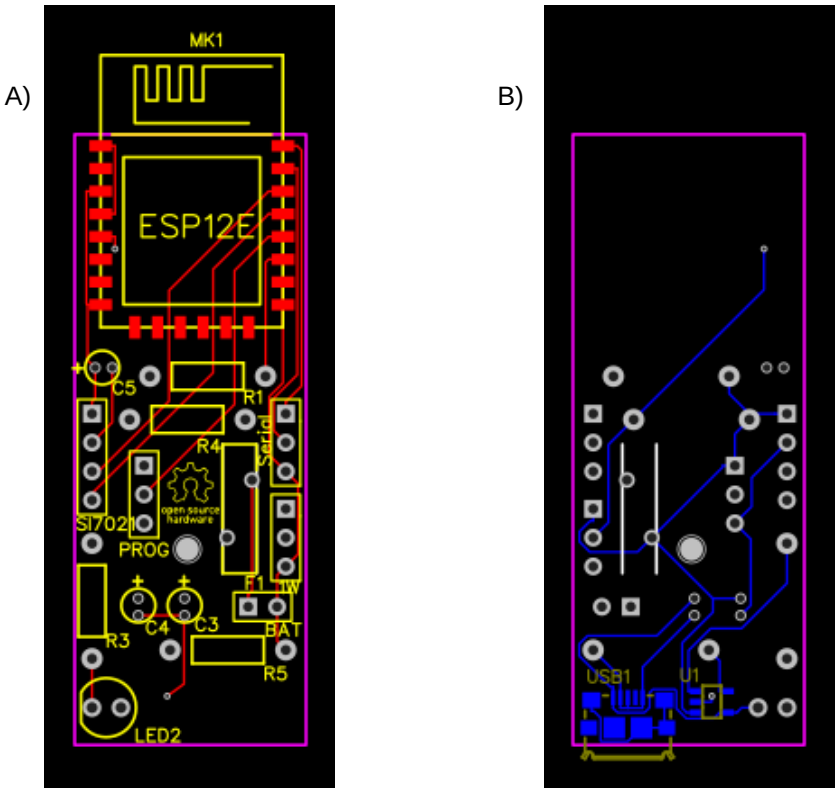
Na schemacie blokowym znajduje się także układ scalony MCP73831 firmy Microchip. Zastosowano standardową aplikację, proponowaną w dokumentacji układu. Pozwala ona na ładowanie podłączonego akumulatora bezpośrednio poprzez zasilanie z portu USB. W celu zabezpieczenia baterii przed zwarciem i w konsekwencji niebezpieczeństwem wystąpienia pożaru, zastosowano bezpiecznik polimerowy.

Celowo zrezygnowano z zastosowania stabilizatora napięcia zasilania, po tym jak eksperymentalnie dowiedziono, że zarówno ESP8266 jak i Si7021 są w stanie w pełni poprawnie pracować nawet przy napięciu zasilania wynoszącym 4.2 V (szczytowe napięcie ładowania akumulatorów litowo-jonowych [14]).

3.1.2.Projekt obwodu drukowanego

Kolejnym krokiem po wykonaniu schematu blokowego było zaprojektowanie prototypu PCB (ang. *printed circut board*, płytka drukowana). Powstały dwie wersje, różniące się rozmiarem i zastosowaną technologią montażu. Ze względu na małoseryjną produkcję i brak dostatecznej umiejętności lutowania u projektanta w technologii SMT (ang. *surface-mount technology*, technologia montażu powierzchniowego*)*, ostatecznie wyprodukowano prototyp o większej powierzchni, z większością elementów montowaną w technologii THT (ang. *through-hole technology*, technologia montażu przewlekanego). Niestety, nie wszystkie elementy posiadały wariant przystosowany do montażu THT. Mikrokontroler ESP8266, układ MCP73831 i gniazdo micro USB zostały zamontowane w technologii SMT.

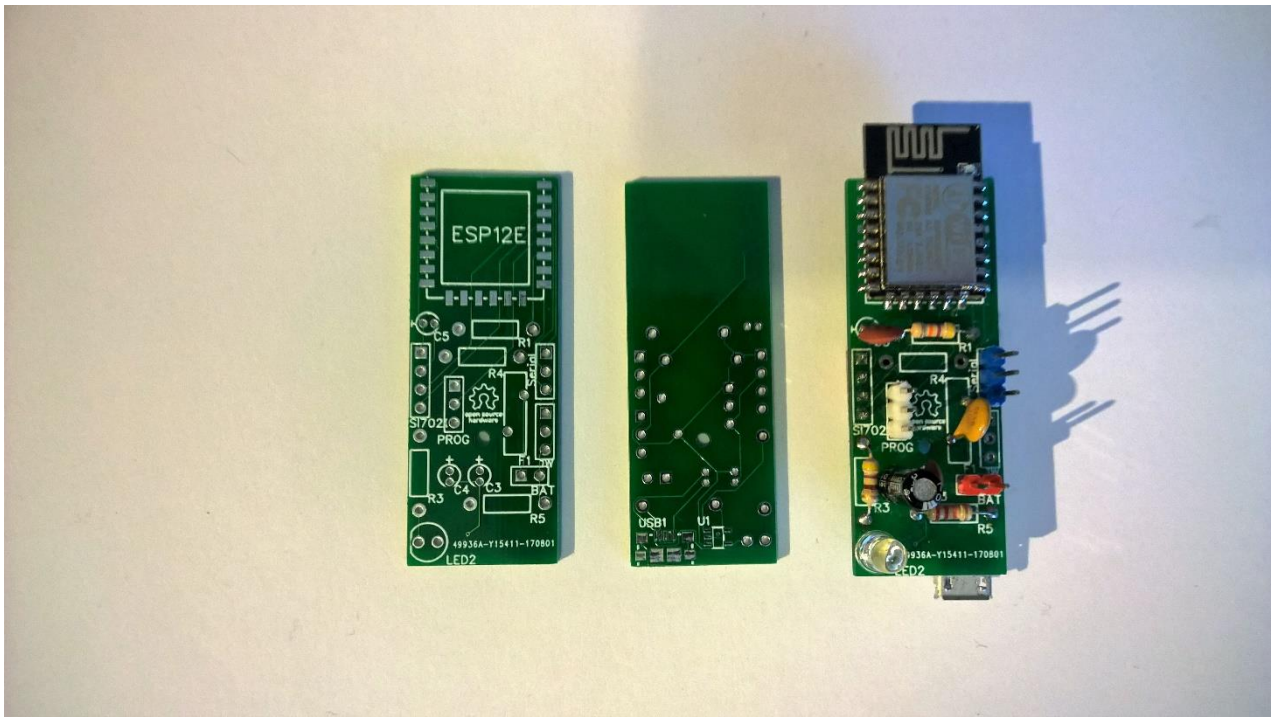
PCB jest dwustronne, raster wyprowadzeń to 2.54 mm (Rys. 3.1). Wymiary płytki z zamontowanymi elementami (20 mm na 60 mm) odpowiadają wymiarom koszyka na jedno ogniwo litowo-jonowe, co pozwala na wygodne ukrycie płytki za baterią.



Rys. 3.1 Projekt PCB: A) strona górna B) strona dolna

3.1.3.Produkcja prototypu

W celu zapewnienia jak najlepszej jakości wykonania zwrócono się do zewnętrznej firmy zajmującej się produkcją obwodów drukowanych. Dzięki temu, możliwe stało się uzyskanie takich detali jak druk dwustronny, metalizacja otworów czy warstwa opisowa, co byłoby bardzo trudne przy samodzielnej produkcji amatorskimi metodami. Na rysunku 3.2 przedstawione zostały puste płytki drukowane i zrealizowany prototyp.



Rys. 3.2 Zrealizowany prototyp wraz z pustymi płytkami drukowanymi

3.1.4. Oprogramowanie ESP8266

Program kontrolujący pracę serwera nie jest skomplikowany, w dużej mierze dzięki zastosowaniu zewnętrznych bibliotek odpowiedzialnych za obsługę WiFi (ESP8266WiFi.h), komunikację z czujnikiem Si7021 (Adafruit_Si7021.h), protokół MQTT (PubSubClient.h) czy też formatowanie wiadomości do JSON (ang. *JavaScript object notation*, lekki format wymiany danych komputerowych, bazujący na podzbiorze języka JavaScript) (ArduinoJson.h). Uproszczony schemat działania programu znajduje się na rys.3.3.

Z racji wykorzystania trybu DeepSleep, program nie wykonuje się w nieskończonej pętli jak jest to zazwyczaj implementowane w mikrokontrolerach, tylko nieustannie resetuje się pod koniec wykonania. Spowodowane jest to implementacją trybu uśpienia w ESP8266. Przy wywołaniu metody deepSleep mikrokontroler wyłącza większość wewnętrznych modułów, pozostawiając tylko uruchomiony zegar, który po określonym czasie zresetuje mikrokontroler poprzez ustawienie pinu IO0 w stan wysoki. Do poprawnego działania tego mechanizmu wymagane jest zewnętrzne podłączenie pinu IO0 do pinu RST mikrokontrolera.

W celu utrzymania jednolitego kodu zarówno do długotrwałych testów pracy na baterii, jak i testów poprawnego wykonania programu zdecydowano się na zastosowanie makr pozwalających na usunięcie z kodu niepotrzebnych fragmentów na etapie preprocesora. W ten sposób, gdy nie zostałoby zdefiniowane marko DEBUG, preprocesor podmieniłby wywołania metod wypisujących dane diagnostyczne poprzez połączenie szeregowe na puste instrukcje. Pozwalało to na zmniejszenie zużycia energii, gdy nie było konieczne przesyłanie informacji diagnostycznych poprzez port szeregowy już na etapie kompilacji programu.

Procesor odczytuje temperaturę i wilgotność z czujnika Si7021 przy pomocy protokołu I^2C. Obsługa protokołu jak i samego odczytu wiadomości jest już zaimplementowana

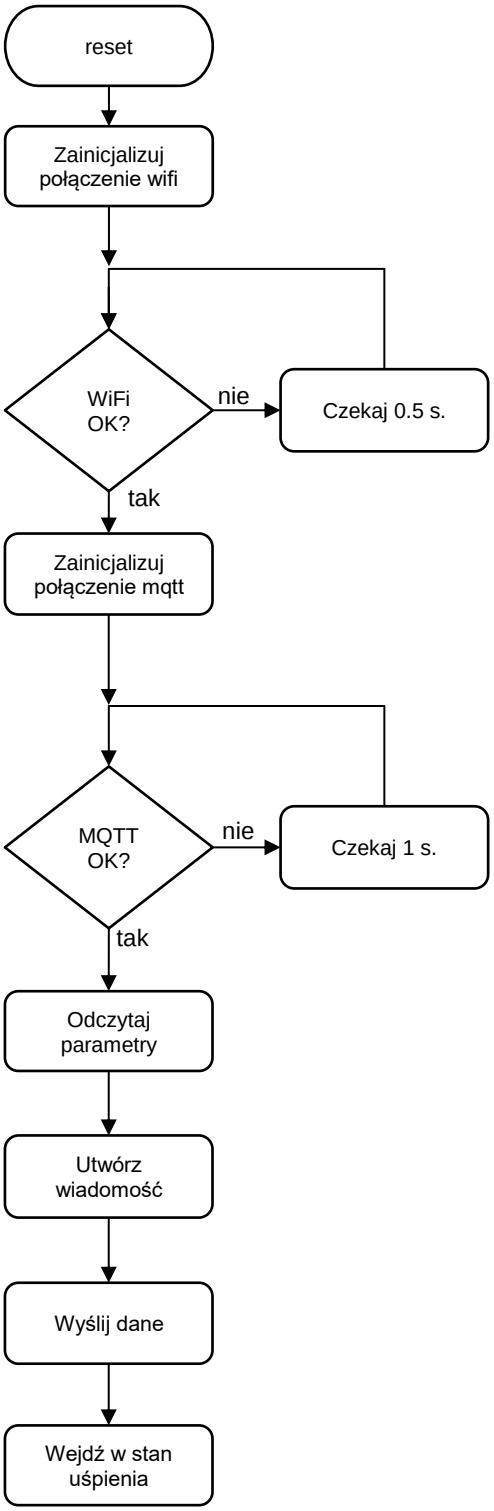
w bibliotekach zewnętrznych. Wskazania czujnika można odczytać przy pomocy dwóch metod: readTemperature i readHumidity.

W ramach projektu odczytywana była także podstawowa telemetria układu. Wykorzystano możliwość wewnętrznego pomiaru napięcia zasilania poprzez jedyny dostępny ADC. Aby zainicjalizować przetwornik wymagane było zastosowanie makra ADC_MODE, w którym należało zadeklarować w jakim trybie będzie on pracował. W celu wykorzystania wewnętrznego połączenia zasilania z ADC, należało skorzystać z trybu ADC_VCC. Odczyt zasilania wykonywała funkcja getVcc.

Poza napięciem zasilania, odczytywana była także ilość pozostałej pamięci. W przypadku tego sensora nie jest to bardzo istotna informacja, jako że układ przechodzi nieustanne resetowanie pamięci w związku z trybem uśpienia. Jednakże, gdyby zaszła potrzeba zastosowania czujnika pracującego w sposób ciągły, odczyt mógłby wskazać wycieki pamięci jeszcze na etapie testowania programu, przed ewentualnym zatrzymaniem działania procesora w wyniku wyczerpania się dostępnej pamięci. Odczyt tego parametru odbywał się w funkcji getFreeHeap.

Jako unikalny identyfikator dla każdego z sensorów wykorzystano jego adres MAC (ang. *medium access control address*). Aby umożliwić generyczne programowanie wielu czujników tym samym oprogramowaniem, skorzystano z odczytu adresu MAC podczas wykonywania programu. Zapewnione było to przez funkcję macAddress, która zwracała adres w formie tablicy sześciu bajtów. W celu późniejszego wykorzystania, adres był konwertowany do tablicy znaków. Całość odbywała się w funkcji setup_wifi.

Według założeń projektowych, wszystkie sensory powinny znajdować się w jednej sieci WiFi. Z tego względu zadecydowano się na umieszczenie danych dostępowych do sieci na stałe w kodzie programu. Jednakże ze względów bezpieczeństwa nie jest to optymalne rozwiązanie. Kwestia bezpieczeństwa została szerzej poruszona w rozdziale 5. (Podsumowanie).



Rys. 3.3 Uproszczony schemat działania programu na mikrokontrolerze

3.2. Aplikacje wspierające

Na aplikację serwerową zasadniczo składają się trzy osobne aplikacje: baza danych MySQL, broker wiadomości MQTT i właściwy serwer obsługujący całość, opisany w podrozdziale 3.3.

3.2.1.Baza danych

W każdej aplikacji zajmującej się przetwarzaniem danych baza danych jest podstawowym komponentem. Ze względu na obiektowy charakter aplikacji głównej, zadecydowano o zastosowaniu ORM (ang. *Object-Relational Mapping*, mapowanie obiektowo-relacyjne). Użyte zostało narzędzie o nazwie SQLAlchemy, które udostępnia mechanizmy pozwalające na mapowanie obiektów do relacji, a także szereg innych operacji ułatwiających korzystanie z języka SQL (ang. *Structured Query Language*, strukturalny język zapytań). Aby w pełni wykorzystać możliwości ORM w aplikacji opartej o framework Flask, użyto rozszerzenia Flask-SqlAlchemy [15]

Na początku prac nad projektem zastosowano bazę danych SQLite, ze względu na łatwość jej użycia i brak zewnętrznych komponentów. Niestety, wraz ze wzrostem ilości danych wymaganych do przetworzenia, wydajność SQLite stała się niewystarczająca. Dzięki wykorzystaniu ORM, zmiana silnika bazy danych na MySQL ograniczyła się jedynie do konfiguracji i migracji danych na nowy system, bez zmian w samym kodzie aplikacji.

Schemat tabeli i relacji w bazie danych zaprezentowano w dodatku F.

3.2.2.Broker MQTT

Jako broker wiadomości MQTT, wykorzystana została aplikacja Mosquitto [11]. Wykorzystanie lokalnego serwera miało wiele korzyści, szczególnie na początku rozwoju projektu, gdy dostęp do logów aplikacji pozwalał na łatwe zdiagnozowanie wszelkich błędów. Razem z aplikacją główną dostarczany jest szereg narzędzi ułatwiających jego wykorzystanie. Szczególnie pomocne okazały się następujące programy: mosquitto_pub i mosquitto_sub które pozwalały na testowanie i podglądanie w czasie rzeczywistym danych wysyłanych przez urządzenia.

Mosquitto pozwala także na szyfrowanie wiadomości z wykorzystaniem protokołu TLS (ang. *transport layer security*, bezpieczeństwo warstwy transportowej). Wykorzystanie tej funkcjonalności byłoby konieczne w przypadku rzeczywistego systemu dostępnego na rynku. W projekcie zrezygnowano z jego implementacji.

3.3. Serwer

Główna aplikacja jest najbardziej rozbudowaną częścią całego projektu. W ramach jednego programu połączona została obsługa nadchodzących wiadomości poprzez sieć MQTT, prezentacja danych pobranych z bazy, kontrola dostępu użytkowników, jak i panel administracyjny. Kod poszczególnych funkcjonalności podzielono na osobne moduły, o ile było

to możliwe. Z racji wykorzystania frameworku Flask, strony WWW były renderowane dynamicznie, przy pomocy silnika szablonów Jinja2. W celu ułatwienia analizy pracy aplikacji i ewentualnego znajdywania błędów zastosowano moduł logów dostępny w bibliotece standardowej języka Python.

3.3.1. Komunikacja z sensorami

Jednym z założeń projektowych była modułowość projektu. W celu jej zapewnienia zdecydowano się na podział kodu obsługującego dane sensory na osobne moduły. Aplikacja przed nadejściem pierwszego żądania HTTP (dekorator *@app.before_first_request*) wywołuje metodę init_devices z pliku models.py. Metoda ta sprawdza zawartość katalogu devices w poszukiwaniu poprawnych modułów języka Python i ładuje je do pamięci. Dzięki temu, rozszerzenie aplikacji o dodatkowe funkcjonalności (np. komunikacja z wykorzystaniem innego protokołu) sprowadza się do napisania odpowiedzialnego za to modułu i zrestartowania serwera. Poprawnie napisany moduł powinien definiować metodę init, która zostanie wywołana po jego załadowaniu do pamięci. Kod metody init_devices został zaprezentowany w Dodatku B.

W celu demonstracji tego mechanizmu napisany został generyczny moduł komunikujący się z sensorami przy pomocy protokołu MQTT i wiadomości kodowanych w formacie JSON. Przy inicjalizacji, moduł subskrybuje odbiór wiadomości w temacie *sensor/JSON* od brokera MQTT. Kiedy wiadomość zostanie odebrana, wywoływana jest metoda handleMQTTMessage (dekorator *@mqtt.on_topic*), w której odbywa się analiza składniowa wiadomości w celu znalezienia wymaganych informacji. Do poprawnego odczytu wiadomości potrzebne są: nazwa typu czujnika (pole *sensor*) i adres MAC (pole *MAC*). W przypadku braku tych pól lub gdy wiadomość nie jest kodowana w UTF-8 serwer odrzuca taką wiadomość jako niepoprawną. W dalszym ciągu szukane są właściwe odczyty czujnika (pola *temp* i *hum*, odpowiednio temperatura i wilgotność), a także telemetria układu. Znalezione dane są następnie zapisywane w bazie danych. W przypadku, gdy adres MAC nie występuje w bazie, tworzony jest nowy wpis do tabeli zawierającej definicje czujników. Kod metody handleMQTTMessage został zaprezentowany w Dodatku C.

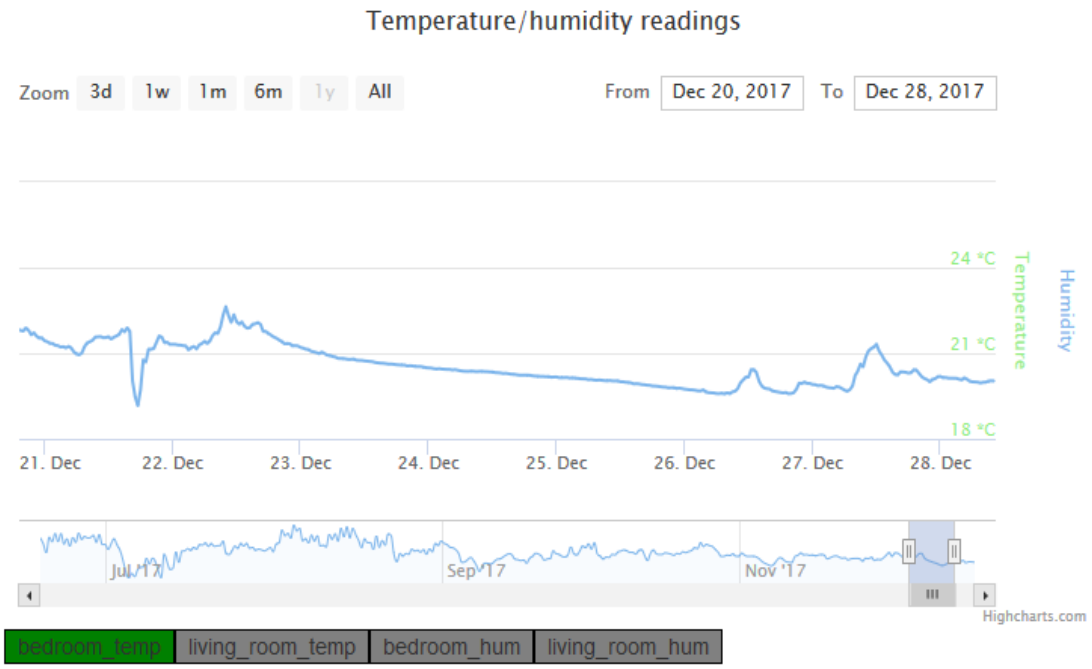
3.3.2. Wyświetlanie danych

Aby cała aplikacja miała zastosowanie praktyczne dla użytkownika wymagana jest prezentacja danych w wygodnej dla niego formie. Sam odczyt wartości liczbowych np. w tabeli nie będzie niósł ze sobą wielu informacji. Dlatego zadecydowano się na prezentację danych przy pomocy wykresu odczytów w osi czasu. W ten sposób, użytkownik jest w stanie natychmiast odczytać pożądane informacje. Przykładowy wykres, ukazujący temperaturę odczytywaną przez jeden z sensorów znajduje się na rys. 3.4. Wyraźnie widać na nim wygładzenie odczytów spowodowane brakiem zmian temperatury, będącym konsekwencją opuszczenia mieszkania w okresie świątecznym.

Tworzenie wykresu odbywa się po stronie użytkownika, przy pomocy języka JavaScript i biblioteki Highcharts.js [16]. Pozwala to na odciążenie serwera, poprzez przeniesienie

obciążającej procesor operacji rysowania wykresu na komputer użytkownika. W początkowej fazie projektu to serwer zajmował się tworzeniem wykresów przy pomocy biblioteki gnuplotlib, ale bardzo szybko okazało się to nieskutecznym rozwiązaniem ze względu na czas trwania całej operacji i brak responsywności (każda zmiana w zapytaniu skutkowała tworzeniem nowego wykresu). W wersji ostatecznej, serwer wysyła tablicę odczytów do użytkownika, który po swojej stronie decyduje o tym co chce zobaczyć. Użytkownik może w ten sposób zmieniać interesujące go odczyty bez oczekiwania na odpowiedź serwera.

Pomimo odciążenia serwera, przesyłanie dużych ilości danych (w chwili pisania pracy, w bazie danych znajduje się ponad sto siedemdziesiąt tysięcy pomiarów) nie jest z punktu widzenia użytkownika natychmiastowe. Aby uniknąć wrażenia powolnego działania strony, przesyłanie danych odbywa się asynchronicznie, z wykorzystaniem biblioteki jQuery i funkcji *getJSON* [17].



Rys. 3.4 Wykres zależności temperatury od czasu z widocznym brakiem zmian w okresie nieobecności domowników.

3.3.3. Kontrola dostępu

Dla zapewnienia podstawowej kontroli nad dostępem do danych, zaimplementowano prosty system kont użytkowników. Użytkownik niezalogowany nie posiada dostępu do żadnej części systemu poza rejestracją. Dane użytkowników zarejestrowanych są przechowywane w bazie danych. Nazwa użytkownika i jego poziom dostępu zapisywane są jawnie, natomiast hasło przechowywane jest w postaci wyniku funkcji skrótu utworzonego przy pomocy algorytmu Bcrypt. Podyktowane jest to względami bezpieczeństwa, gdyż nawet w przypadku niepowołanego dostępu do bazy danych, atakujący nie będzie w stanie w trywialny sposób odczytać hasła użytkownika ze względu na specyfikę algorytmu. Bcrypt, w przeciwieństwie do większości algorytmów skrótu, działa wolno [18], co uniemożliwia stworzenie słownika haseł w rozsądnym czasie. Implementację tego algorytmu w języku Python zapewnia biblioteka py-bcrypt,

a funkcje potrzebne dla frameworka Flask dodaje pakiet flask-bcrypt. Implementacja logowania znajduje się w Dodatku D.

Po zalogowaniu się, uzyskuje się dostęp do odczytów sensorów na podstronie *Show all*. Użytkownik ma także dostęp do edycji i podglądu detali swojego konta na podstronie *Profile*.

3.3.4. Panel administracyjny

Nie każdy zarejestrowany użytkownik ma dostęp do panelu administracyjnego. Jedynie użytkownicy z ustalonym poziomem uprawnień w kolumnie *level* w tabeli użytkowników są w stanie korzystać z funkcjonalności tam dostępnej. Administrator jest w stanie zmieniać poziom dostępu innych użytkowników, a także sprawdzać detale ich kont oraz zmieniać nazwy konkretnych sensorów i odczytywać ich właściwości.

Odnośniki do panelu administracyjnego są wyświetlane tylko wtedy, gdy użytkownik ma do nich dostęp. Odpowiedzialny za to jest silnik szablonów Jijna2 i odpowiedni kod umieszczony w szablonie strony WWW. Przykład wykorzystania znaczników jest przedstawiony w Dodatku E. Widać tam polecenia udostępniane przez silnik szablonów takie jak struktury kontrolne *{% if %}* czy też bezpośrednie wywołanie kodu, w postaci wywołania funkcji *url_for*.

4. Wyniki testowania

W ramach testowania projektu, poza testami funkcjonalności skupiono się na sprawdzeniu zapotrzebowania na energię sensorów.

4.1. Testy funkcjonalne

Podczas tworzenia kodu aplikacji, wszystkie funkcjonalności były testowane na bieżąco. Niestety, nie było to rozwiązanie optymalne, ponieważ brak odgórnego rozplanowania aplikacji doprowadził do łańcuchów zależności, które utrudniały analizę i rozwój aplikacji. Niemniej, wszystkie funkcje zostały przetestowane i działają poprawnie. W celu poprawy czytelności UI (ang. *user interface*, interfejs użytkownika) zwrócono się do osób niebędących związanymi z projektem o ocenę wyglądu aplikacji.

4.2. Uruchamianie prototypu sensora

Przed zamówieniem obwodów drukowanych, schemat blokowy prototypu sensora został przetestowany na płytce prototypowej. Sprawdzanie parametrów układu takich jak napięcie czy prąd pobierany w stanie uśpienia odbyło się z wykorzystaniem miernika UNI-T UT58C. Testowanie mikrokontrolera ESP8266 i układu MCP72831 wykonano z użyciem oscyloskopu cyfrowego DSO-2090. Nie napotkano poważnych trudności na tym etapie prac.

4.3. Pomiar zapotrzebowania na energię

W celu zbadania zapotrzebowania na energię, mierzono długość czasu pracy układu od pełnego naładowania akumulatora do zadziałania wewnętrznych systemów odcinających zasilanie, chroniących akumulatory przed nadmiernym rozładowaniem.

Zbadane zostały dwa źródła energii, ogniwo litowo-jonowe firmy Panasonic o pojemności 3400 mAh (układ A) i akumulator litowo-polimerowy firmy Akyga o pojemności 980 mAh (układ B). W ramach testów, obydwa układy działały z identycznymi wersjami oprogramowania i dokonywały odczytu co 5 minut. Napięcie zasilania mierzone było przez wewnętrzny ADC. W związku z większą pojemnością, układ A pracował nieprzerwanie od 7.07.2017 aż do 18.12.2017, natomiast układ B w tym samym czasie był trzykrotnie ładowany. Przez cały czas trwania pomiarów, baterie znajdowały się w temperaturze pokojowej. Należy wziąć to pod uwagę w przypadku projektowania sensorów mających działać poza domem, ponieważ niska temperatura przyśpieszy rozładowanie akumulatora.

Z rysunku 4.5 odczytać można wskazania ADC od 25.08.2017r. do dnia 18.12.2017r., kiedy nastąpiło odcięcie zasilania. Niestety, brak jest danych na temat całego cyklu pracy, ponieważ pomiar napięcia zasilania został wprowadzony dopiero 25.08, a sensor działał już od 7.07. Interesująca jest nieliniowość odczytów, może ona wskazywać na niedokładność działania przetwornika analogowego wbudowanego w ESP8266.

Na rysunku 4.6 doskonale widać cały cykl pracy, od ładowania w dniu 18.09 do rozładowania i odcięcia zasilania w dniu 6.11. Na szczególną uwagę zasługuje zmiana tempa spadku napięcia po przekroczeniu 3.5 V. To wskazanie jest świetnym ostrzeżeniem dla użytkownika o konieczności wymiany lub naładowania baterii.

Uzyskana długość czasu pracy na zasilaniu bateryjnym jest szczególnie satysfakcjonująca w przypadku układu A. Osiągnięty czas pracy wynoszący prawie pół roku, stawia pod znakiem zapytania sens korzystania z wbudowanego układu do ładowania akumulatora, jeżeli byłaby to czynność tak rzadko wykonywana. Zwiększając czas uśpienia układu do pół godziny, można wydłużyć czas pracy nie tracąc wiele na dokładności pomiarów.

Rys. 4.5 Wykres zależności napięcia w czasie dla układu A (ogniwo litowo-jonowe)

Rys. 4.6 Wykres zależności napięcia w czasie dla układu B (ogniwo litowo-polimerowe)

5. Podsumowanie

Praca nad projektem była niewątpliwie cennym doświadczeniem, które zaowocuje w przyszłości. Poruszone zostały tematy z odległych dziedzin informatyki, jakimi są systemy wbudowane, bazy danych, czy też projektowanie stron WWW. Jednakże w chwili obecnej projekt nie jest niczym więcej niż prototypem, w celu udostępnienia takiego rozwiązania na rynku potrzeba jeszcze wiele pracy.

Poziom zabezpieczeń w aplikacji jest podstawowy, nieadekwatny do wymagań rynkowych. Brak szyfrowanego połączenia MQTT naraża użytkowników na podsłuchanie informacji wysyłanych przez sensory. Ponadto, w chwili obecnej serwer akceptuje wszystkie pozornie poprawne dane, nie sprawdzając uwierzytelnienia czujnika. W ten sposób, atakujący mógłby podszyć się pod dowolny sensor i wysłać na serwer fałszywe dane nie będąc wykryty. Równie istotny jest nieszyfrowany protokół HTTP. W dzisiejszych czasach, szyfrowanie ruchu www w sieci WAN (ang. *wide area network*, rozległa sieć komputerowa) jest niemalże standardem, w niedalekiej przyszłości takie same wymagania będą stawiane serwisom działającym w sieciach LAN.

Nie mniej ważny od bezpieczeństwa jest też próg opłacalności takich rozwiązań. Małoseryjna produkcja prostych technicznie sensorów nie jest w stanie konkurować z gigantami produkcyjnymi znajdującymi się w Shenzhen. Materiały potrzebne na wykonanie pięciu sensorów wykorzystanych w projekcie kosztowały ok. 100$. Oprócz tego doliczyć należy czas potrzebny na zaprojektowanie, złożenie i przetestowanie gotowego produktu. W tym samym czasie, w sklepach internetowych można znaleźć już gotowe rozwiązania oferujące nawet większą funkcjonalność za ok. 10$ [19].

Bibliografia

[1] *. J. O. 2. Carles Gomez 1, „Overview and Evaluation of Bluetooth Low Energy:An Emerging Low-Power Wireless Technology,” *Sensors,* pp. 8-9, 2012.

[2] Espressif, „ESP8266EX Datasheet,” 2017. [Online]. Available: http://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. [Data uzyskania dostępu: 15 01 2018].

[3] I. Grokhotkov, 2017. [Online]. Available: http://arduino-esp8266.readthedocs.io/en/latest/reference.html. [Data uzyskania dostępu: 15 01 2018].

[4] I. Grokhotkov, „Minimal ESP8266 hardware setup,” 2107. [Online]. Available: http://arduino-esp8266.readthedocs.io/en/latest/boards.html#minimal-hardware-setup-for-bootloading-and-usage. [Data uzyskania dostępu: 15 1 2018].

[5] Postscapes, 2017. [Online]. Available: https://www.postscapes.com/internet-of-things-protocols/. [Data uzyskania dostępu: 15 01 2018].

[6] R. N. T. Roy T. Fielding, „Principled Design of the Modern,” *ACM Transactions on Internet Technology,* tom 2, nr 2, pp. 115-150, 2002.

[7] F. e. al., „Hypertext Transfer Protocol -- HTTP/1.1,” *Standards Track,* 1999.

[8] OASIS, „Oasis Open,” 10 12 2015. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf. [Data uzyskania dostępu: 15 01 2018].

[9] A. Ronacher, „Flask,” 2018. [Online]. Available: http://flask.pocoo.org/. [Data uzyskania dostępu: 15 01 2018].

[10] D. Plentz, „MQTT Brokers,” [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki/servers. [Data uzyskania dostępu: 15 01 2018].

[11] Eclipse, „Mosquitto,” 2018. [Online]. Available: https://mosquitto.org/. [Data uzyskania dostępu: 15 1 2018].

[12] Oracle, „MySQL,” 2018. [Online]. Available: https://www.mysql.com/. [Data uzyskania dostępu: 15 01 2018].

[13] Silicon Labs, „Si7021-A20,” 8 2016. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf. [Data uzyskania dostępu: 15 01 2018].

[14] Mircochip, „MCP73831,” 2008. [Online]. Available: https://www.sparkfun.com/datasheets/Prototyping/Batteries/MCP73831T.pdf. [Data uzyskania dostępu: 15 01 2018].

[15] A. Ronacher, „FlaskSqlalchemy,” 2017. [Online]. Available: http://flask-sqlalchemy.pocoo.org/2.3/. [Data uzyskania dostępu: 15 01 2018].

[16] Highcharts, „Master-detail chart,” 2018. [Online]. Available: https://www.highcharts.com/demo/dynamic-master-detail. [Data uzyskania dostępu: 15 01 2018].

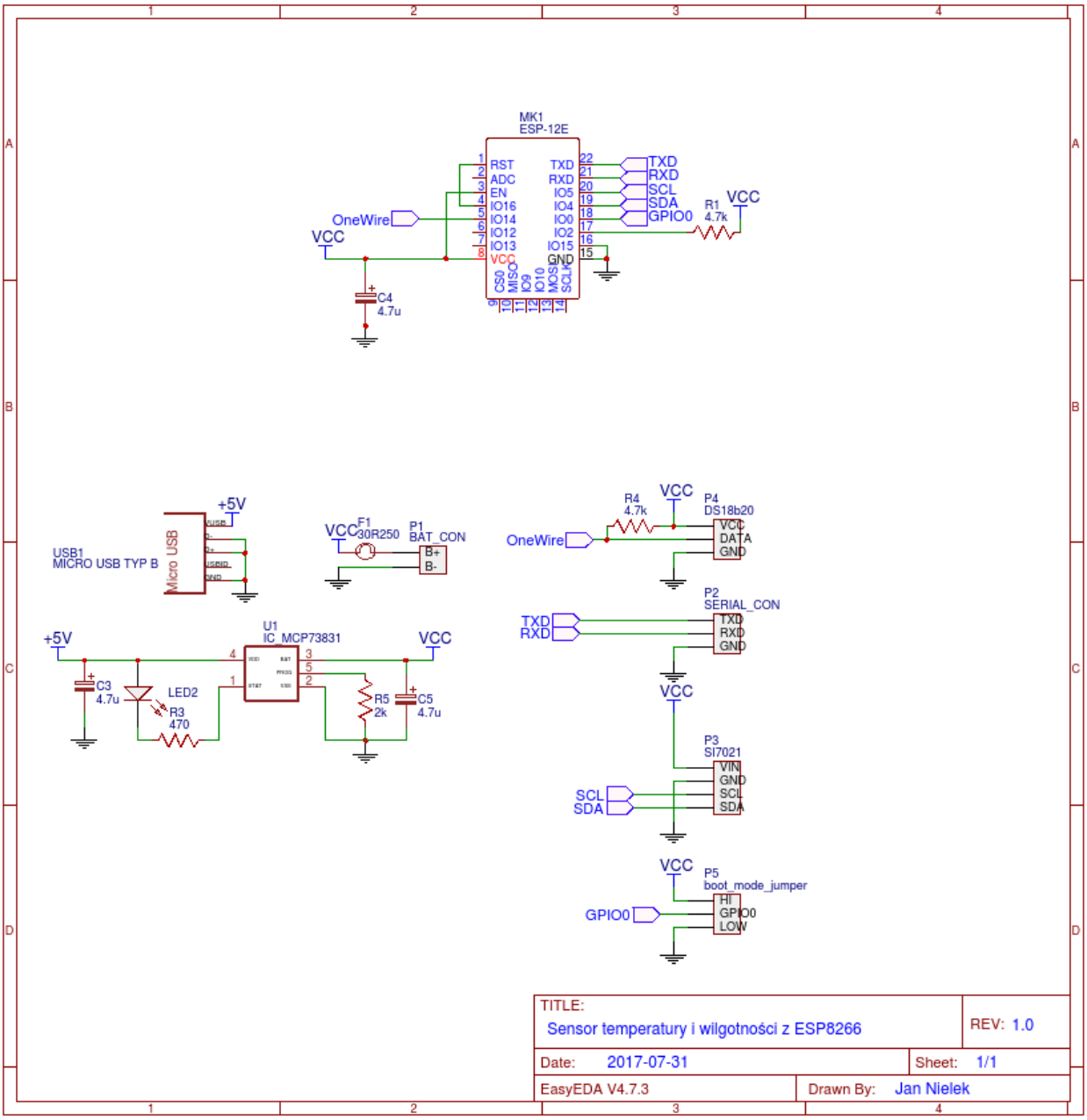
[17] jQuery Fundation, „jQuery API reference,” 2018. [Online]. Available: http://api.jquery.com/jquery.getjson/. [Data uzyskania dostępu: 15 01 2018].

[18] D. M. Niels Provos, „Bcrypt evaluation,” 28 4 1999. [Online]. Available: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node11.html#SECTION00062000000000000000. [Data uzyskania dostępu: 15 01 2018].

[19] Hyelesiontek, „New ESP8266 NODEMCU 0.96-inch Preflashed Expansion Temperture And Humidity Development Board Weather Station,” 2018. [Online]. Available: https://www.aliexpress.com/item/New-ESP8266-NODEMCU-0-96-inch-Preflashed-Expansion-Temperture-And-Humidity-Development-Board-Weather-Station/32847973221.html?spm=2114.search0104.3.64.1c7999566KanNl&ws_ab_test=searchweb0_0,searchweb201602_4_10152_10151_100. [Data uzyskania dostępu: 15 01 2018].

Spis rysunków

Dodatek A: Schemat ideowy sensora temperatury



Dodatek B: Kod metody obsługującej ładowanie modułów

```
@app.before_first_request
def init_devices():
    import pkgutil
    import importlib
    modules = [name for _, name, _ in pkgutil.iter_modules(['FUSS/devices'])]
    logging.info('Got {} module(s)'.format(len(modules)))
    devices = []
    for m in modules:
        logging.info('Loading device module {}'.format(m))
        moduleName = 'FUSS.devices.{}'.format(m)
        devModule = importlib.import_module(moduleName)
        devices.append(m)
        app.register_blueprint(devModule.device_blueprint, url_prefix=('/dev/' + m))
        devModule.init()
    app.config.update(dict(DEVICES=devices))
```

Dodatek C: Kod metody obsługującej wiadomości MQTT

```
@mqtt.on_topic(device_topic)
def handleMQTTMessage(client, userdata, message):
    message = message.payload
    logging.info('got {}'.format(message))
    try:
        msg = str(message, 'utf-8')
        msg = json.loads(message)
    except ValueError as err:
        logging.ERROR('Failed to parse json message! Error: "{}"\nMessage: "{}"'.
                      format(err, message))
        return
    if 'sensor' not in msg or 'MAC' not in msg:
        logging.ERROR('JSON lacks required fields! {}'.format(msg))
        return
    devType = msg['sensor']
    try:
        mac = int(msg['MAC'], 16)
    except ValueError as err:
        logging.ERROR('Could not parse MAC address. {}'.format(err))
        return
    with app.app_context():
        if 'temp' in msg:
            func_id = 1
            insert_data(devType, mac, func_id, float(msg['temp']))
        if 'hum' in msg:
            func_id = 2
            insert_data(devType, mac, func_id, float(msg['hum'])).id
        if 'vcc' in msg and 'heap' in msg:
            t = SensorTelemetry(mac, date=datetime.now(),
voltage=float(msg['vcc'])/1000.0, free_heap=msg['heap'], json_data=message)
            db.session.add(t)
            db.session.commit()
```

Dodatek D: Kod metody logującej użytkownika

```
def login(name, password):
    user = User.query.filter_by(username=name).first()
    if (user is None or bcrypt.check_password_hash(user.password, password) == False):
        return False, 'Invalid username or password'
    session['logged_in'] = True
    session['isAdmin'] = user.isAdmin()
    session['userID'] = user.id
    return True, 'Logged in'
```

Dodatek E: Fragment szablonu strony głównej

```
<div id="navbar" class="navbar-collapse collapse">
    <ul class="nav navbar-nav navbar-right">
        <li><a href="#">Dashboard</a></li>
        {% if not session.logged_in %}
            <li><a href="{{ url_for('register_user') }}">Register</a></li>
            <li><a href="{{ url_for('login') }}">log in</a></li>
        {% else %}
            {% if session.isAdmin %}
                <li><a href="{{ url_for('admin_panel') }}">Admin</a></li>
            {% endif %}
            <li><a href="{{ url_for('view_all')}}">Show all</a></li>
            <li><a href="{{ url_for('profile_panel') }}">Profile</a></li>
            <li><a href="{{ url_for('logout') }}">log out</a></li>
        {% endif %}
    </ul>
</div>
```

Dodatek F: Schemat relacyjnej bazy danych