

POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRONIKI I TELEKOMUNIKACJI



PRACA DYPLOMOWA MAGISTERSKA

**SYSTEM INTELIGENTNEGO STEROWANIA
LISTWAMI ZASILANIA**

Wojciech Krzysztof Olszewski

Promotor:

Dr inż. Krzysztof Arnold

Poznań, 2019

Streszczenie

Tematem pracy jest projekt bezprzewodowego systemu do inteligentnego sterowania listwami zasilania, czyli zarządzania gniazdami w przedłużaczach 230V w sposób automatyczny lub poprzez telefon/komputer.

Praca obejmuje 8 rozdziałów, w których zostały opisane wykorzystane platformy, projekt i oprogramowanie sieci, scenariusze użytkowania, historia jej uruchamiania oraz przykładowe rozwiązania dostępne na rynku. Na końcu można znaleźć podsumowanie oraz spis dodatków ze schematami.

Autor stara się dość skrupulatnie i zrozumiale wtajemniczyć czytelnika w jego sposób myślenia i działanie systemu.

Praca nie zawiera pełnego kodu programu i strony internetowej, co wynika z jego obszerności. Dostępne są one w osobnej części, wydrukowanej jako dokument uzupełniający do pracy.

Spis treści:

Streszczenie	3
Wstęp	7
1. Charakterystyka mikrokomputera Raspberry Pi 3	9
1.1. Architektura.....	9
1.2. Podstawowe parametry	10
1.3. Środowisko programistyczne	10
2. Charakterystyka mikroprocesora Arduino Nano	11
2.1. Architektura.....	11
2.2. Podstawowe parametry	12
2.3. Środowisko programistyczne	12
3. Projekt systemu sterowania listwami zasilającymi.....	13
3.1. Założenia projektowe	13
3.2. Koncepcja systemu.....	13
3.2.1. Moduły systemu i wybór elementów	13
3.2.2. Schemat systemu.....	15
3.2.3. Podział zadań	17
3.3. Opis systemu	18
4. Oprogramowanie systemu	21
4.1. Koncepcja oprogramowania.....	21
4.2. Projekt bazy danych	23
4.3. Oprogramowanie mikrokomputera Raspberry Pi	29
4.3.1. Protokół transmisji.....	29
4.3.2. Program zarządzania siecią.....	30
4.3.3. Program do wizualizacji	35
4.3.4. Program menadżera zadań	36
4.4. Oprogramowanie platformy Arduino Nano	37
4.5. Oprogramowanie strony internetowej.....	38
4.5.1. Logowanie, uprawnienia.....	38
4.5.2. Panel główny.....	40
4.5.3. Panel wizualizacji	42
4.5.4. Panel zadań automatycznych	43
4.5.5. Panel dziennika gniazd	44
4.5.6. Panel konfiguracyjny	45
4.5.7. Wersja mobilna strony	48

5.	Sterowanie listwami	53
6.	Uruchamianie systemu	57
6.1.	Środowisko Raspian	57
6.2.	Środowisko stacji roboczej	58
6.3.	Problemy związane z systemem	59
6.3.1.	Problem transmisji pakietów	59
6.3.2.	Problem uruchamiania systemu operacyjnego	62
6.3.3.	Problem dostępowy (Access Point)	62
6.3.4.	Problem sterowania mocą	62
6.3.5.	Problem zadań kontrolowanych czasowo	63
6.3.6.	Problem utraty połączenia	63
6.4.	Etapy prac nad projektem	63
6.5.	Testowanie systemu	68
7.	Przegląd podobnych rozwiązań	71
8.	Podsumowanie	75
	Bibliografia	77
	Dodatek A Schemat prototypu listwy	79
	Dodatek B PCB prototypu układu	81
	Dodatek C Fragment noty katalogowej NRF24L01	83
	Dodatek D Wykaz elementów	85

Wstęp

Pomysł stworzenia projektu na bazie Raspberry Pi zrodził się nie z przypadku. Autor zaangażował się w projekt grantowy, w którym mikrokomputerem miało być właśnie to urządzenie. Już pierwsze spotkanie stworzyło okazję do sprawdzenia mocy obliczeniowej urządzenia oraz jego możliwości wraz z wykorzystaniem wejść/wyjść GPIO (ang. General-purpose Input/Output, pol. wejść/wyjść ogólnego przeznaczenia). Wiedza autora z zakresu środowisk UNIX, była na ten moment jedynie podstawowa, ale wiadome było, że system Raspian (dystrybucja Debiana stworzona dla RPi) stwarza możliwości, które warto wykorzystać w poważniejszym projekcie. Kluczowe znaczenie miał fakt, że nie było tutaj systemu czasu rzeczywistego, a system z wywłaszczaniem na którym uruchomić można serwer hostujący stronę WWW, bazę danych oraz, co ważne, wykorzystać piny GPIO do własnych celów. Autor tworzył w przeszłości strony internetowe mniej lub bardziej zaawansowane, wykorzystywał bazy danych, a także pisał programy w językach wysokiego poziomu, takich jak C (na mikrokontrolerach) oraz C++. Co więcej, miał także szeroką wiedzę z zakresu telekomunikacji.

Praca magisterska powinna być ambitna, a proponowane w niej rozwiązania przemyślane i przydatne dla potencjalnego użytkownika. Ma ona być zwieńczeniem 5 lat studiów i dobrze byłoby, gdyby czas poświęcony na jej realizację zaowocował budzącymi zadowolenie wynikami. Efektem takiego podejścia, jest pomysł stworzenia systemu bezprzewodowego, przeznaczonego do automatycznego i ręcznego sterowania listwami zasilania. Przed rozpoczęciem projektowania przeanalizowano podobne rozwiązania, podejmując decyzję, że opracowanie urządzeń będzie tworzone głównie na podstawie własnego podejścia.

Tematem pracy jest System Inteligentnego Sterowania Listwami Zasilania. Określenie inteligentny należy rozumieć tutaj jako automatyzację działania systemu, a więc ograniczenia interakcji systemu z użytkownikiem do minimum po pełnej automatyzacji procesu. Projekt polega na opracowaniu systemu modułów dołączanych do listw zasilania przedłużaczy 230V oraz zapewnieniu jego działania od strony programowej. W każdym z gniazd użyte będą czujniki, aby można było gniazda sterować za ich pomocą. Sterowanie odbywać się będzie poprzez stronę WWW w wersji komputerowej i mobilnej, na smartfonie, z zabezpieczoną domową siecią lokalną oraz panelem logowania z uprawnieniami. Na stronie będzie można załączać i wyłączać gniazda, ustawiać automatyczne zadania oraz sprawdzić zapisy z dziennika zmian stanów. Dostępny będzie panel konfiguracyjny do wprowadzania zmian w systemie oraz szczegółowe dane z czujników w postaci czasowych wykresów z wyborem przedziału czasu.

Celem projektu jest wprowadzenie do domu możliwości zdalnego zarządzania wybranymi urządzeniami poprzez ich automatyzację lub ręczną kontrolę. System pozwoli także na kontrolę temperatury w okolicach listw zasilających oraz sprawdzenie, jak zmieniało się natężenie światła w ciągu np. ostatniej doby. Zadania automatyczne mają zwolnić użytkownika z włączania oświetlenia w momencie, gdy w pomieszczeniu bądź na zewnątrz robi się ciemno lub uruchamiania grzejnika elektrycznego, gdy temperatura spadnie poniżej zadanego poziomu.

1. Charakterystyka mikrokomputera Raspberry Pi 3

Programy dla mikrokontrolera Raspberry Pi (rys. 1.1) mogą być tworzone w prosty sposób, w miarę możliwości z podziałem na konkretne funkcje lub na osobne procesy. Raspberry Pi posiada system operacyjny, który pozwala na działanie wielu procesów równolegle w tle poprzez odpowiednie mechanizmy (wywłaszczanie). Projekt będzie się opierał głównie na uruchomionych w nieskończonej pętli programach, które mogą funkcjonować niezależnie w tym samym czasie.



Rys. 1.1 Wygląd RPi 3 B

Wybrany język programowania dla RPi (Raspberry Pi, RasPi lub malinka) to C++. Wynika to z dość dobrej wiedzy autora na jego temat oraz tego, że mikrokomputer natywnie pozwala na kompilacje takich programów.

Opis techniczny RasPi zostanie przedstawiony w podrozdziałach 1.1 i 1.2.

1.1. Architektura

Obecnie istnieje wiele wersji popularnej malinki. Różnią się one peryferiami, specyfikacją, budową, wymiarami i wieloma innymi aspektami.

Docelowo system miałby pracować na najbardziej minimalistycznej wersji, czyli Raspberry Pi Zero, która to jest najtańsza ze wszystkich oraz najmniejsza. Zasoby jakimi ona dysponuje także są szczuplejsze, porównywalne z wersją 1, która posiada tylko 1 rdzeń i 1 GB RAM, ale nadaje się do naszego celu.

Wersja prototypowa, w pracy dalej omawiana, to Raspberry Pi 3 B. Jest ona prawie najmocniejszą wersją RPi na rynku, minimalnie słabszą od wersji 3 B+. Posiada wszystkie do tej pory instalowane wyjścia jak i prędkość, która będzie potrzebna do testów systemu.

Główne elementy mikrokomputera to [1]:

- Układ SoC (ang. System on a Chip) – zawiera w jednej strukturze (układzie scalonym) procesor, pamięć oraz procesor graficzny, który notabene przetwarza także dźwięk. Układy te różnią się w zależności od wersji, ale są to układy firmy Broadcom bazujące na 64-bitowych procesorach z rodziny ARM. Są one niskonapięciowe (3,3V), aby były energooszczędne i wydzielaly mniej ciepła.
- Pamięci:

- DRAM – układy zawierają ulotną pamięć RAM o pojemności od 256 MB do 1024 MB, w zależności od wersji. Jest ona scalona z układem SoC w jednym module oraz zasilana niskonapięciowo.
- SD Flash – trwała pamięć do przechowywania systemu operacyjnego oraz plików użytkownika. Są pewne ograniczenia sterownika co do wielkości takich pamięci. Do projektów wystarczają pamięci do 16 GB.
- Złącza – RPi posiada cały wachlarz złącz, do których można podłączyć wszelkiego rodzaju peryferia, a są to:
 - HDMI – gniazdo pozwalające podłączyć monitor, może być także w wersji mini (RPi Zero),
 - kompozytowe wyjście wideo – służy ono do wyświetlania obrazu na analogowych monitorach,
 - audio – gniazdo na wtyki mini-jack do wydobywania dźwięku,
 - USB – mamy do dyspozycji kilka złącz USB, np. aby podłączyć mysz i klawiaturę, bądź wszelkiego rodzaju adaptery, np. USB-Ethernet (RPi Zero).
 - złącza rozszerzeń – mowa tutaj o złączach CSI i DSI, do podłączania wyświetlacza dotykowego oraz kamery,
 - GPIO – złącze wejścia/wyjścia ogólnego zastosowania, które obsługuje różne protokoły, np. SPI, UART, itd.,
 - złącze zasilania – układ zasilamy poprzez port Micro-USB,
 - Ethernet – port służący do podłączenia kabla sieciowego RJ-45.

1.2. Podstawowe parametry

Podstawowymi parametrami Raspberry Pi 3 B są [2]:

- Czworordzeniowy procesor Broadcom BCM2837 64bit taktowany zegarem 1.2GHz,
- 1GB pamięci RAM,
- Moduł BCM43438 dla bezprzewodowej sieci LAN oraz chip Bluetooth Low Energy (BLE),
- 10/100 Mb/s złącze ethernetowe,
- 40 pinów na listwie GPIO,
- 4 porty USB w wersji 2,
- 4 stykowe wyjście audio i kompozytowe,
- pełnowymiarowe wyjście HDMI w wersji 1.3a,
- 15 pinowy port CSI do podłączenia kamery,
- 15 pinowy port DSI do podłączenia ekranu dotykowego,
- gniazdo microSD obsługujące karty klasy 10,
- wejście MicroUSB o maksymalnym poborze prądu 2,5A.

1.3. Środowisko programistyczne

Środowiskiem programistycznym będzie program Microsoft Visual Code, który pozwala na łatwe programowanie poprzez szybkie przemieszczanie się po kodzie, kolorowanie składni czy podgląd funkcji. Jest on programem windowsowym, zainstalowanym na stacji roboczej. Kod następnie będzie synchronizowany poprzez program WinSCP oraz protokół SCP, z programem na RPi. W ostatnim kroku kod będzie budowany za pomocą polecenia g++, które kompiluje pliki cpp do wyjściowych programów wykonawczych.

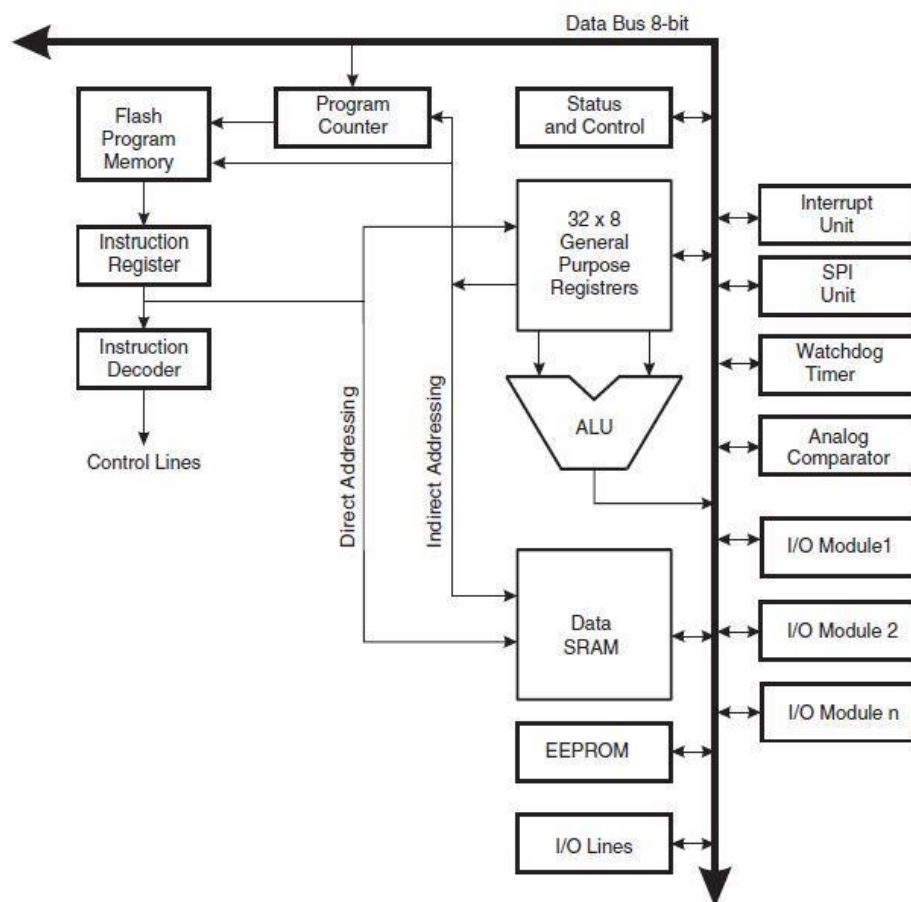
2. Charakterystyka mikroprocesora Arduino Nano

Platforma Arduino jest bardzo popularna wśród entuzjastów prostych projektów elektronicznych. W szczególności nadaje się do prototypowania, z racji ceny, stabilności oraz bazy bibliotek. Językiem programowania jest w ogólności język *Wiring*, który jednak bazuje na języku C++ [3]. Należy pamiętać także, że platforma jak i środowisko programistyczne jest Open-Source'owe, każdy może oprogramowanie modyfikować i je usprawniać. Tworzy to dodatkowe możliwości, które pierwotnie były niedostępne.

2.1. Architektura

Architektura platformy różni się w zależności od modelu oraz wersji. Do tego momentu stworzono co najmniej 20 jej różnych odmian [4]. Większość z nich posiada złącze USB do programowania oraz zasilania układu. Aby można było za jego pośrednictwem programować Arduino, na płycie drukowanej musi być zamontowany programator. Istnieją wersje, gdzie trzeba dołączyć do pinów GPIO dodatkowy układ, który będzie w stanie wgrać napisany przez nas program do pamięci trwałej układu.

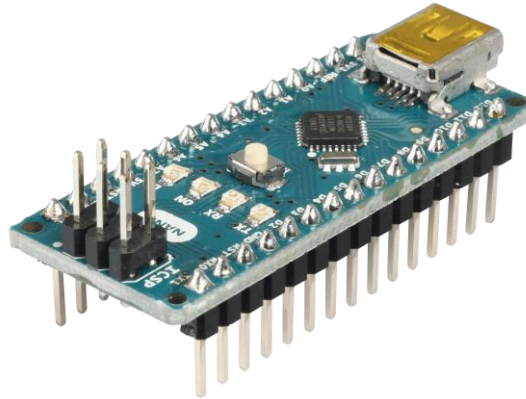
Sam mikrokontroler to głównie 8-bitowy układ Atmega, do którego podłączony jest wcześniej wspomniany programator, diody informujące, zegar kwarcowy, listwa z wyjściami ogólnego zastosowania, przycisk RESET. Uogólnioną architekturę kontrolerów z rodziny AVR, czyli także Atmega przedstawia rysunek 2.1 [5].



Rys. 2.1 Architektura AVR [5]

2.2. Podstawowe parametry

Projekt listwy w wersji prototypowej, roboczej nazywanej *IPACS* (Intelligent Power Adapter Control System) bazować będzie na replice Arduino Nano. Wybór tego właśnie modułu wynika z tego, że jest on tańszy niż inne wersje, posiada wbudowany programator ze złączem MiniUSB oraz jest mały. Wygląd tego modułu przedstawiono na rysunku 2.2.



Rys. 2.2 Wygląd Arduino Nano [5]

Podstawowe parametry wersji Nano 3.3 to [28]:

- mikrokontroler Atmega328,
- napięcie pracy 5V,
- 32 KB pamięć flash, z których 2 KB są wykorzystywane na bootloader,
- 2 KB pamięci SRAM,
- 1 KB pamięć EEPROM,
- zegar 16 MHz,
- 8 pinów analogowych (przetworniki ADC 10-bitowe),
- 22 pinów cyfrowych (6 z obsługą PWM),
- interfejsy komunikacyjne UART, I2C, SPI,
- zewnętrzne przerwania,
- maksymalny prąd na pin 40 mA,
- napięcie wejściowe 7-12V,
- pobór prądu 19 mA.

2.3. Środowisko programistyczne

Środowiskiem programistycznym jest stacja robocza z systemem Windows 10 oraz oprogramowanie Arduino IDE, które może przypominać prosty notatnik z możliwością kompilacji i wgrywania kodu oraz sprawdzania składni. Tworzy ono poza wiedzą użytkownika projekt, który zawiera wszystkie niezbędne biblioteki oraz kod. To co widzi użytkownik to, podstawowy szablon projektu w języku *Wiring*, który jest maksymalnie uproszczony.

3. Projekt systemu sterowania listwami zasilającymi

3.1. Założenia projektowe

Przed postawieniem sobie celów oraz poświęceniem czasu na analizę, jak miałby system działać należało przeanalizować rozwiązania już istniejące.

Naczelnym założeniem projektu jest jego funkcjonowanie bezprzewodowe oraz możliwość automatyzacji zleconych systemowi zadań.

Pozostałe założenia to:

- możliwość sterowania listwami zasilającymi 230V,
- niezależne włączanie/wyłączanie każdego gniazda w listwie,
- możliwość przemieszczania listw w obrębie mieszkania lub domu,
- wykorzystanie bazy danych do gromadzenia danych,
- prosty w obsłudze interfejs webowy,
- panel logowania oraz uprawnienia dla użytkowników,
- dobowe statystyki dla użytkownika,
- rejestrowanie zmian w systemie,
- panel do automatyzacji zadań,
- panel konfiguracyjny systemu,
- możliwość rozbudowy systemu, np. poprzez zwiększenie liczby listw.

Wymagania stawiane systemowi nie są zbyt wygórowane, biorąc pod uwagę doświadczenie i kreatywność autora. System w miarę jego rozwijania z pewnością otrzyma kilka dodatkowych możliwości, które sprawią, że będzie bardziej funkcjonalny i przyjazny dla użytkownika.

W projekcie celem nadrzędnym jest prostota użytkowania oraz wprowadzenie funkcji użytecznych. Skomplikowanie systemu oraz jego rozbudowę o kolejne elementy możemy pozostawić na inny czas. System będzie konstruowany tak, aby możliwa była jego znaczna modyfikacja, w celu jego rozwoju.

3.2. Koncepcja systemu

3.2.1. Moduły systemu i wybór elementów

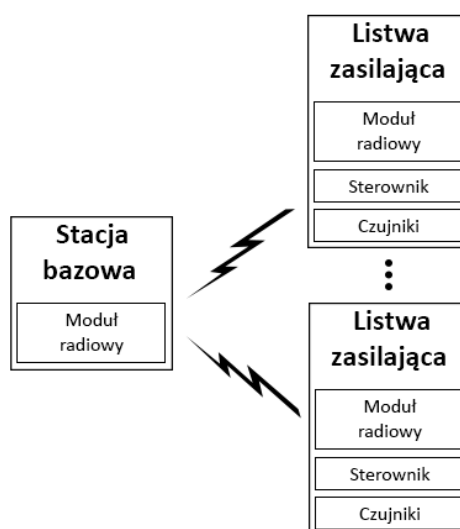
Przed wyborem elementów systemu (rys. 3.1) należy się dobrze zastanowić, gdyż od tego zależy, jak projekt będzie się kształtował i jaka będzie jego sprawność.

Pod uwagę trzeba wziąć:

- **moduły bezprzewodowe** - na rynku jest wiele urządzeń pozwalających odbierać i nadawać dane bezprzewodowo, jednak część z nich jest albo droga (WiFi) albo ograniczona zasięgiem (Bluetooth) albo po prostu pozbawiona sprzętowej kontroli transmisji (moduły RF433 MHz). Istnieją też takie, które są kompromisem pomiędzy ceną a parametrami. W projekcie zostanie użyty moduł NRF24L01, a raczej jego odpowiednik z chińskiego rynku, który cechuje się częstotliwością pracy 2,4 GHz (pasmo ISM), sterowalną mocą, retransmisjami,

CRC. Więcej informacji na jego temat zawarte jest we fragmencie noty na końcu pracy (Dodatek C).

- **czujniki** - system ma funkcjonować automatycznie bazując między innymi na czujnikach. Najbardziej odpowiednie wydają się być sensor temperatury oraz natężenia światła z racji tego, że zazwyczaj chcemy coś uruchamiać za pomocą mierzonych przez nie wielkości fizycznych i wydają się być wystarczającym wyborem dla projektu. Wybrano model DS18B20 [6], czyli cyfrowy czujnik temperatury z dokładnością pomiaru $\pm 0,5\text{ }^{\circ}\text{C}$ i rozdzielczością maksymalnie 12 bitów oraz analogowy fotorezystor [7] jako sensor natężenia światła, który należy użyć w układzie dzielnika napięcia, aby odczytać z niego dane.
- **język programowania** - nie ma zbytniego wyboru jeśli chodzi o budowanie kodu na platformę Arduino, możemy wybierać między C a jego pokrewną odmianą, językiem Wiring, który jest dedykowanym językiem na tą platformę, bądź też w Assemblerze, co jednak z racji prostoty funkcji wydaje się być zbędne. Wybór środowiska i języka padł na dedykowane platformie Arduino IDE i Wiring. Raspberry natomiast możemy programować w języku Python, który nie jest znany autorowi, dlatego też wybrał on język C++. Strona internetowa będzie wykorzystywać głównie PHP do skryptów, CSS do obsługi graficznej, HTML do szkieletu strony oraz JavaScript dla bardziej zaawansowanych skryptów. Baza danych będzie obsługiwana przez język skryptowy SQL.
- **protokół transmisji** - część protokołu transmisji zapewnia sam układ NRF24L01 [8], który sprzętowo zabezpiecza ramki cyklicznym kodem nadmiarowym CRC, retransmisjami oraz informacją zwrotną ACK o poprawnym odebraniu pakietu. Nie pozwala to jednak na monitorowanie połączenia, czy jest ono słabe lub czy zasięg jest słaby. Wymagane jest tym samym zaprogramowanie takiego mechanizmu z poziomu systemu.



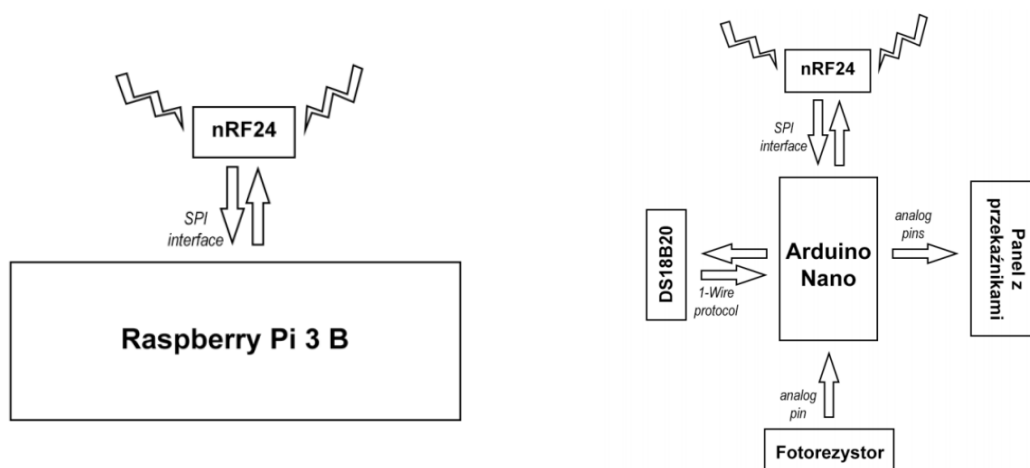
Rys. 3.1 Ogólny schemat systemu

- **bezpieczeństwo systemu** - tutaj trzeba przemyśleć jakie będą dostateczne normy bezpieczeństwa dla takiego systemu, który bądź co bądź, będzie pozwalał uruchamiać poszczególne urządzenia w domu. Użycie tego systemu przez osobę nieupoważnioną pozwoli wygenerować dodatkowe koszty lub uprzykrzyć życie.

Autor doszedł do wniosku, że stworzenie systemu za zaporą sieciową sieci lokalnej z nieznanym dla osoby obcej IP oraz panelem logowania, który dodatkowo sprawdza uprawnienia osoby logującej i generuje logi, jest wystarczające.

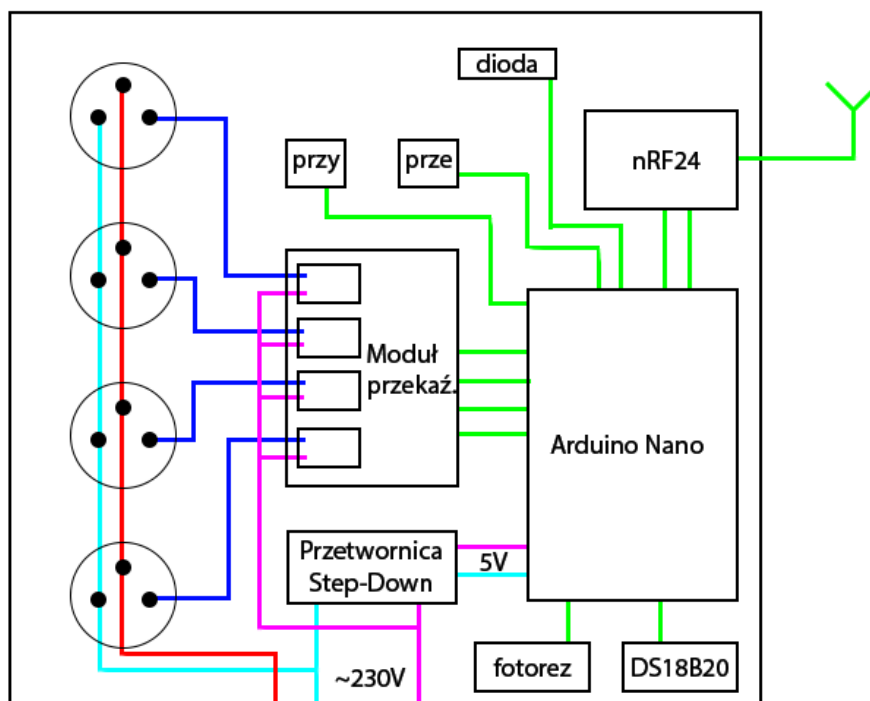
3.2.2. Schemat systemu

W ogólności system możemy opisać poprzez układ modułów łączących się przez radio ze stacją bazą (matką) Raspberry Pi, która zarządza pracą całego systemu (rys. 3.2)



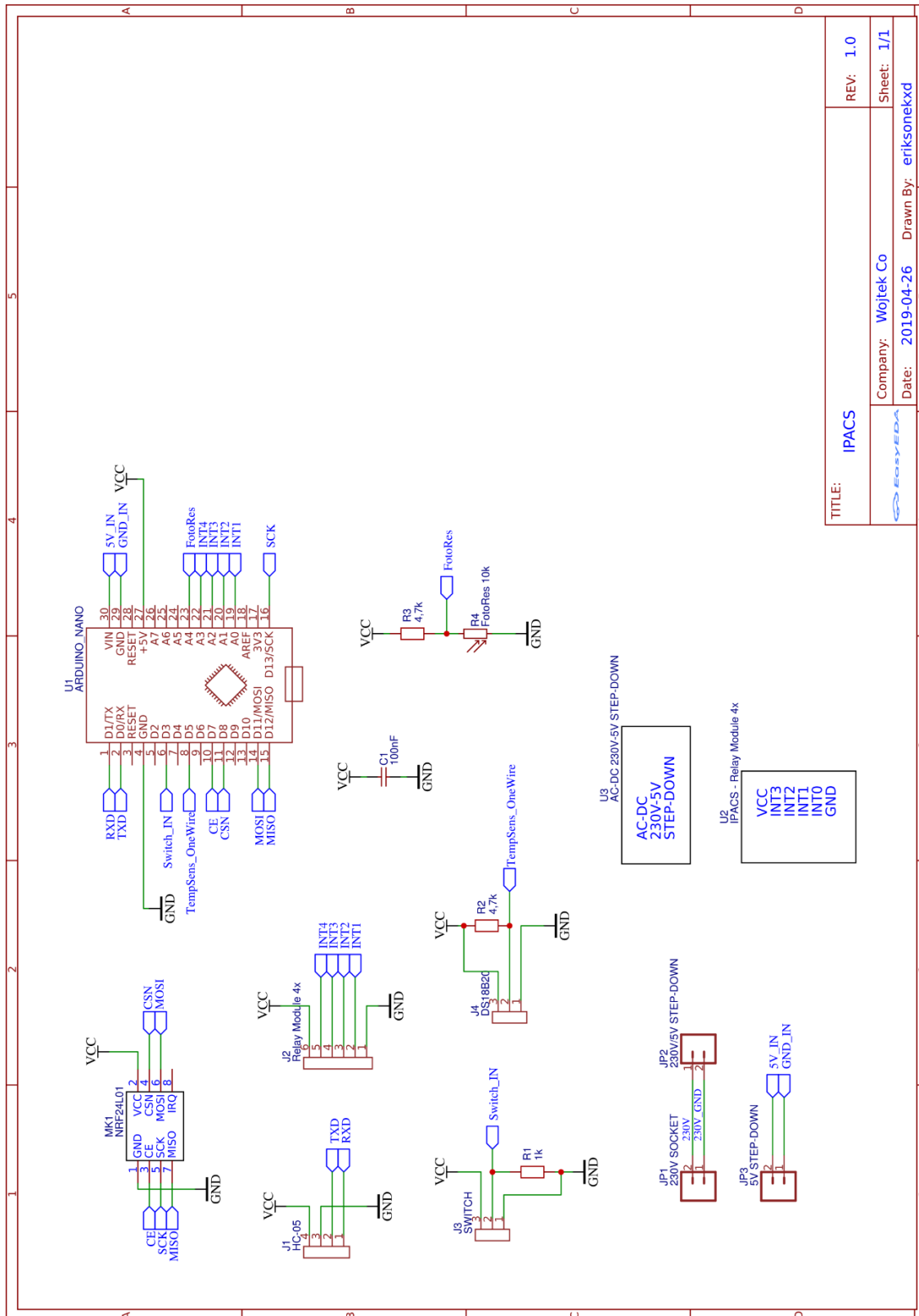
Rys. 3.2 Schemat systemu

Uogólnioną architekturę modułu listwy przedstawia rysunek 3.3.



Rys. 3.3 Uogólniony schemat moduły listwy

Nie pokazuje on interfejsów oraz rzeczywistych połączeń odpowiednich komponentów, gdyż ma on jedynie pokazać koncepcję. Projekt szczegółowy oraz końcowy przedstawiony jest na rysunku 3.4.



Rys. 3.4 Szczegółowy schemat moduły listwy

3.2.3. Podział zadań

Zadania w pracy należy podzielić na te, które dotyczą koncepcji systemu i projektu sprzętowego oraz na zadania oprogramowania, które są opisane poprzez wykonywane funkcje programu zarządzającego.

Do zadań sprzętu należy:

- pomiar temperatury i natężenia oświetlenia,
- załączanie wybranych gniazd z wykorzystaniem przekaźników,
- sterowanie przekaźnikami listwy z wyprowadzeń modułu Arduino,
- bezprzewodowe sterowanie systemem listw z poziomu nadrzędnego mikrokomputera Raspberry Pi, z wykorzystaniem modułów radiowych.

Do zadań programu należy:

- odczyt wyników pomiaru temperatury i pomiaru natężenia oświetlenia (poziom Arduino),
- przysyłanie wyników pomiarów do mikrokomputera Raspberry Pi (poziom Arduino),
- odbiór poleceń z Raspberry Pi (poziom Arduino),
- zbieranie danych w bazie danych,
- prezentacja danych na stronie WWW,
- obsługa poleceń za pośrednictwem strony WWW,
- automatyczna realizacja zadań zaprogramowanych za pośrednictwem strony WWW.

Realizacja systemu wymaga wykonania przedstawionych poniżej zadań.

Zadania projektowo-uruchomieniowe to:

- złożenie układu testowego na płytce stykowej (inaczej zwanej montażową),
- uzyskanie testowego połączenia pomiędzy Raspberry Pi a Arduino Nano poprzez moduły NRF24L01,
- odczytanie danych z czujników temperatury oraz natężenia światła,
- napisanie programu do wysyłania danych zebranych z czujników,
- uzyskanie komunikacji dwukierunkowej,
- skonfigurowanie Raspberry Pi, aby można było hostować serwer webowy z PHP oraz bazy danych,
- stworzenie odpowiedniej bazy danych,
- napisanie programu do zarządzania danymi w bazie danych,
- napisanie strony internetowej do wizualizacji wyników,
- oprogramowanie strony, aby była możliwość sterowania gniazdami poprzez bazę danych oraz wizualizację danych z czujników,
- utworzenie panelu konfiguracyjnego do wprowadzania zmian do systemu bez potrzeby ręcznej ingerencji w bazę danych,
- napisanie programu oraz podstrony do obsługi menadżera zadań automatycznych,
- dodanie panelu logowania i uprawnień w systemie,
- utworzenie dziennika zmian stanów gniazd poprzez ręczną zmianę bądź automatyczną.

Zadania związane z oprogramowaniem RPi:

- napisanie programu głównego, który odbiera i nadaje wiadomości oraz sprawdza odpowiednie parametry połączenia, a jeśli zachodzi potrzeba wykonuje odpowiednie kroki w postaci wyłączenia obsługi listwy bądź zmian mocy listw. Program ma pobierać dane z bazy danych i je tam przechowywać oraz weryfikować parametry adapterów i systemu co pętlę wykonania programu,
- napisanie programu do obsługi menadżera zadań, który ma działać niezależnie od programu głównego i sprawdzać zaiscie warunków zadania zdefiniowanego przez użytkownika w stałym odstępie czasu, sprawdzając poprawność podanych warunków,
- napisanie programu do zbierania danych z czujników z odstępem czasu 15 minut o stałych porach czasowych; odczytu danych z bazy danych każdej włączonej listwy z włączonym czujnikiem i przesłanie ich do odpowiedniej tabeli w bazie.

Zadania dotyczące projektu będą szerzej opisane w rozdziale 6, gdzie zajmiemy się uruchamianiem systemu oraz etapami jego budowania. Kolejny rozdział natomiast zostanie poświęcony opisowi oprogramowania i jego poszczególnych funkcji, w tym problemów z nimi związanych.

3.3.Opis systemu

Działanie systemu będzie polegać na podziale na funkcje. Raspberry jako stacja matka pełnić będzie funkcję Master i będzie odpytywać, przy użyciu odpowiedniego protokołu transmisji, listwy, które są w stanie obsługi przez system. Stan obsługi to stan, w którym listwa jest sterowalna przez stronę, w momencie braku połączenia listwa automatycznie przechodzi w stan „nie obsługiwana”, fizycznie nie wyłączając się przy tym, zachowując stan sprzed utraty połączenia. Odpytywanie będzie można także wyłączyć bądź włączyć ręcznie z poziomu panelu konfiguracyjnego. Listwy zasilające, nazywane także adapterami, będą pełnić funkcję Slave i będą czekać na żądanie ze stacji matki. W wiadomości adresowanej do listwy będzie stan gniazd w listwie bądź inna komenda konfiguracyjna. Oba elementy systemu będą musiały rozpoznać zaadresowaną wiadomość poprzez adresowanie od 00 do 99, gdzie pierwszy adres to adres RPi. Daje nam to 99 możliwych slave’ów do zaadresowania w systemie.

System przy użyciu modułów bezprzewodowych z antenami typu Unipol na pasmo 2,4 GHz powinien działać z zasięgiem zbliżonym do sieci Wi-Fi. Zbliżonym dlatego, że urządzenia Wi-Fi dysponują większą mocą, szerszym pasmem oraz lepszym algorytmem dostępu do medium CSMA/CA. Moduły wykorzystane w projekcie bazują na modulacji GFSK, która polega na kształtowaniu impulsu krzywą gaussowską oraz na zmianie częstotliwości przy zmianie bitu. Dodatkowo w przypadku, gdy wiadomość ACK (Acknowledgement, potwierdzenie odbioru) nie przyjdzie do nadajnika, to wysyłana jest ponownie wiadomość, która czeka w buforze tak długo aż osiągnie limit czasu bądź przyjdzie ACK. Nie pozwala to jednak na zweryfikowanie, za którym razem wiadomość doszła do adresata oraz czy są jakieś problemy w transmisji.

Należy zaprogramować protokół do kontroli transmisji, to znaczy do sprawdzania czy każde z urządzeń odbiera wiadomości lub w jakim stopniu dochodzi do zaników sygnału. Może to oznaczać, że należy taką listwę przesunąć bądź zmienić jej moc nadawczą.

Ten sam protokół musi też zapobiegać przed kolizją wiadomości, gdyż platformy Arduino w różnych momentach czasu odbierają i nadają. Raspberry dodatkowo nie jest systemem czasu rzeczywistego, gdyż posiada system operacyjny z wywłaszczaniem, który priorytetyzuje procesy. Może się zdarzyć, że komendy do listw zostaną wysłane w czasie, który spowoduje kolizję. Zegar RPi to co najmniej 1 GHz, czyli RPi działa około 60x szybciej niż przetwarza dane układ adaptera.

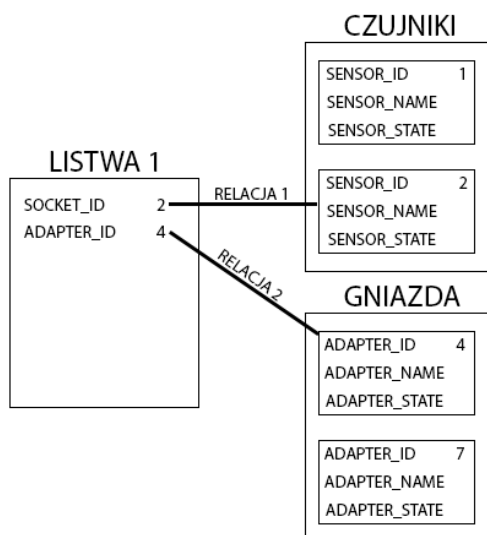
Każdy adapter będzie miał możliwość, w przypadku rozłączenia systemu lub braku połączenia, na przejście w sterowanie ręczne, które załączy wszystkie przekaźniki ignorując stany gniazd przychodzące ze stacji matki jako żądanie. W takim przypadku raport z czujników oraz sygnał o tym, że listwa jest sterowana ręcznie zostaje nadany zwrotnie w celu przekazania informacji użytkownikom na stronie internetowej.

4. Oprogramowanie systemu

4.1. Koncepcja oprogramowania

Projekt ma charakter ewolucyjny, nie ma zdefiniowanych granic, co wynika z faktu, iż platforma Raspberry ma olbrzymie możliwości. Oprogramowanie musi być budowane umożliwiając dalszy rozwój projektu. Rodzaj projektu wymaga od autora rozpoczęcia pracy nad oprogramowaniem od strony komunikacji pomiędzy listwą a stacją bazową. Gdy to już będzie osiągnięte, prace można kontynuować dowolnie według podziału zadań przedstawionego w rozdziale 3.2.3, gdzie opisane zostały cele stawiane projektowi oraz oprogramowaniu.

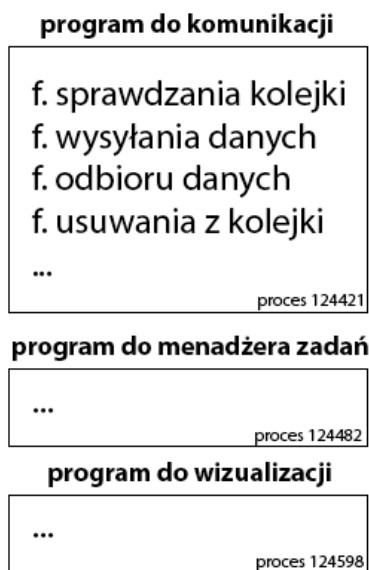
Projektowanie rozpoczęte zostanie od bazy danych, bo to w niej będą przechowywane dane i na niej opierać się będzie fundament sieci. Można ją porównać do terminala komunikacyjnego, w którym informacje są bardzo często zapisywane i odczytywane. Koncepcję takiej relacyjnej bazy przedstawia rysunek 4.1. Kolejnym logicznym następstwem będzie opisanie programu zarządzającego całym systemem, który składać się będzie z 3 głównych programów. Program z kontrolą komunikacji natomiast składać się będzie z szeregu funkcji (rys. 4.2). Zbiór instrukcji przewidzianych dla procesora ATmega w języku *Wiring* przedstawia rysunek 4.3. Ostatnim elementem, jednak bardzo istotnym, będzie interfejs użytkownika w postaci strony internetowej. Będzie to trudne zadanie, gdyż musi być on zaprojektowany tak, aby był prosty i łatwy w obsłudze, a jednocześnie był ładny oraz funkcjonalny. Pomysł interfejsu został pokazany na rysunku 4.4.



Rys. 4.1 Koncepcja relacyjnej bazy danych

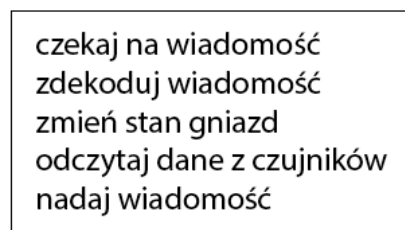
Tworzenie bazy danych nie należy do prostych zadań. Mamy do dyspozycji programowanie komendowe lub graficzne. Przy tym pierwszym trudność jest taka, że wcześniej trzeba sobie projekt rozpisnąć i kontrolować składnię języka. Z tego powodu użyty

będzie, po doinstalowaniu, dodatek *phpMyAdmin*. Służy on do zarządzania bazami danych i jest powszechnie używany. Można za jego pomocą łatwo wizualizować tabele oraz relacje.



Rys. 4.2 Koncepcja programu zarządzania

Program do zarządzania systemem nie będzie miał formy graficznej, a to co się dzieje w systemie nie będzie nigdzie przedstawiane, dlatego konieczne będzie dodanie przejrzystego trybu do logowania wszelkich ważniejszych zmian, które będą zachodzić na bieżąco. Utrudnieniem będzie brak graficznego debuggera, który jest dostępny np. przy programowaniu w Visual Studio. Istnieje możliwość podpięcia się zdalnie do Raspberry poprzez ten program, który można kompilować na stacji roboczej, lecz trudności powstaną przy dołączaniu bibliotek, które trzeba zainstalować z poziomu mikrokontrolera. Sporo informacji o błędach i ostrzeżeń zapewni kompilator, ale bez logów nie zweryfikujemy poprawności działania samego systemu. To samo dotyczy platformy Arduino.



Rys. 4.3 Koncepcja instrukcji na platformę Arduino

Tworzenie strony polega na tworzeniu szkieletu wraz z wymaganymi skryptami do zarządzania treścią zwanego *backend* oraz na modyfikacji wizualnej tego szkieletu zwanego *frontend*. Strona musi być stworzona tak, aby możliwie łatwo można było modyfikować tak skrypty, jak i część graficzną. Trudno powiedzieć, która część będzie wymagać więcej wkładu czasowego, bo projekt powtarzając słowa ma charakter ewolucyjny. W trakcie tworzenia wiele fragmentów będzie modyfikowanych i zmienianych. Z pewnością jednak można powiedzieć, że szata graficzna przez to, że musi być dopasowana do treści będzie poprawiana dużo częściej niż pozostałe elementy z racji tego, że musi być spójna.



Rys. 4.4 Koncepcja wyglądu strony internetowej

4.2. Projekt bazy danych

Baza danych w projekcie opierać się będzie na systemie MariaDB [9], który został stworzony przez pierwotnych twórców oryginalnego MySQL [10]. Z różnych powodów podjęto decyzję o stworzeniu nowego rodzaju baz danych na licencjach typu open-source z wsteczną kompatybilnością z MySQL. Zaletą tego rozwiązania jest to, że twórcy mają bliski kontakt ze społecznością a ta ma wpływ na rozwój projektu. Dzięki takiemu podejściu MariaDB zyskuje dużą popularność.

Przy instalacji bazy danych na systemie Raspian nie mamy jednak możliwości instalacji bazy MySQL, prawdopodobnie z powodów licencji. Dlatego wybór padł na jej wcześniej opisany w pełni darmowy odpowiednik.

Bezpieczeństwo zapewnione jest poprzez sieć lokalną oraz system logowania z uprawnieniami do konkretnych baz danych oraz do konkretnych operacji na nich. Standardowe wymagania dla haseł użytkowników phpMyAdmin oraz MariaDB są dość silne, a najlepiej sprawdzają się te wygenerowane automatycznie przez stronę.

Baza danych będzie nosić nazwę *ipacs_database*. Aby otrzymać klarowność i porządek nazwy tabel będą dosłownie oznaczać ich przeznaczenie, tj. dla przykładu tabela, gdzie znajdują się informacje i parametry listw będzie nosić nazwę *adapters*. Zestawienie nazw tabel oraz ich przeznaczenie znajdziemy w tabeli 4.1.

Nazwa tabeli	Przeznaczenie
adapters	spis listw wraz z parametrami
sockets	spis gniazd wraz z parametrami
sensors	spis czujników wraz z parametrami
socket_logs	logi zawierające informacje o gniazdach
socket_tasks	zadania zdefiniowane przez użytkowników
sensor_data	dane z sensorów zbierane co 15 min
users	spis użytkowników wraz z informacjami
job_list	spis zmian konfiguracji zleconych systemowi
system_settings	ustawienia systemu
stats	statystyki odbioru pakietów

Tabela 4.1 Spis tabel w bazie ipacs_database

Aby dobrze zrozumieć bazę danych sieci trzeba każdą z tabel opisać:

- Encja¹ *adapters* służy do przetrzymywania podstawowych danych na temat listw takich jak nazwa (pokój i lokalizacja), innych ważnych parametrów dotyczących ich działania jak i połączenia. Ich opis przedstawia tabela 4.2, która jest zrzutem ekranu z interfejsu systemu zarządzania bazą danych, *phpMyAdmin*.

#	Nazwa	Typ	Metoda porównywania napisów	Atrybuty	Null	Ustawienia domyślne	Komentarze	Dodatkowo
1	adapter_id 	int(11)			Nie	Brak		AUTO_INCREMENT
2	adapter_room	text	utf8mb4_general_ci		Nie	Brak		
3	adapter_location	text	utf8mb4_general_ci		Nie	Brak		
4	adapter_state	tinyint(1)			Nie	0		
5	adapter_website_control	tinyint(1)			Nie	1		
6	adapter_removed	tinyint(1)			Nie	0		
7	adapter_connection	smallint(1)			Tak	1		
8	adapter_powerLevel	text	utf8mb4_general_ci		Nie	Brak		
9	adapter_waitNumber	int(11)			Nie	60		

Tabela 4.2 Przykładowy spis atrybutów encji *adapters* (rzeczywisty zrzut)

Wyjaśniając atrybuty, *typ* oznacza typ przechowywanej zmiennej, mogą one być różne od tekstowego, po liczbowe z określonymi długościami łańcucha znaków bądź bitowymi, później mamy *metodę porównywania napisów*, która jest wykorzystywana przy zmiennych tekstowych i pomaga dekodować znaki, następnie wykorzystywane przez nas są *ustawienia domyślne*, które wymuszają na rekordzie² jego konkretną wartość w momencie uzupełniania danymi, gdy sami jej nie wpisujemy oraz *dodatkowe*, gdzie np. w tej sytuacji występuje *auto_increment*, co oznacza, że nie

¹ Encja – jest to reprezentacja obiektu bazy danych, która ma określoną strukturę i może, ale nie musi mieć zapisanych w nim danych, inaczej zwana jest dla uproszczenia tabelą. [11]

² Rekord – w języku baz danych, obiekt stanowiący pewną zdefiniowaną strukturę mogący istnieć samodzielnie bądź jako element innego rekordu, w ogólności jest to zapis w bazie danych. [12]

musimy pamiętać, który element wpisujemy z kolei, gdyż wartość atrybutu zostanie automatycznie zwiększona o jeden względem wartości ostatniego rekordu.

Przy kolejnych encjach autor zrezygnował ze szczegółowego spisu atrybutów, gdyż duża część kolumn i ich ustawień będzie się powtarzać w sposób analogiczny. Z opisu zadań, który przedstawia tabela 4.3 można w łatwy sposób wywnioskować, jak takie atrybuty będą wyglądać od strony strukturalnej.

Nazwa atrybutu	Zadanie
adapter_id	unikalny identyfikator danej listwy, klucz główny tabeli ³
adapter_room	nazwa pomieszczenia, w którym znajduje się listwa
adapter_location	miejsce, lokalizacja listwy w pomieszczeniu
adapter_state	stan listwy (włączona lub wyłączona z obsługi)
adapter_website_control	status sterowania listwą (przez stronę bądź ręczny)
adapter_removed	flaga usunięcia listwy z systemu
adapter_connection	stan połączenia z listwą
adapter_powerLevel	wybrana, potwierdzona moc listwy
adapter_waitNumber	liczba prób połączenia z listwą
adapter_weak_signal	próg słabego sygnału
adapter_beacon	włączenie stałego odpytywania

Tabela 4.3 Opis atrybutów tabeli *adapters*

- Encja *sockets* służy do przechowywania danych na temat gniazd wraz z odnośnikiem do adaptera, do którego gniazdo należy. Opis zadań przedstawia tabela 4.4. Będzie ona często odczytywana w celu wysłania stanu gniazd do odpowiedniego adaptera.

Nazwa atrybutu	Zadanie
socket_id	unikalny identyfikator gniazda, klucz główny
adapter_id	identyfikator listwy, do której gniazdo należy
socket_name	nazwa gniazda zdefiniowana przez użytkownika
socket_state	stan gniazda (włączone lub wyłączone)
socket_task_control	flaga kontroli gniazda przez zadanie automatyczne

Tabela 4.4 Opis atrybutów tabeli *sockets*

- Encja *sensors* służy do przechowywania informacji na temat sensorów oraz do bieżących danych z sensorów otrzymanych drogą radiową. Ta tabela będzie najczęściej aktualizowana przez program. Jej opis przedstawia tabela 4.5.

Nazwa atrybutu	Zadanie
----------------	---------

³ Klucz główny, podstawowy (primary key) – niepowtarzalny atrybut tabeli, który jednoznacznie identyfikuje według jakiego atrybutu encja jest indeksowana oraz tworzona jest relacja. [13]

sensor_id	unikalny identyfikator czujnika, klucz główny
adapter_id	identyfikator listwy, do której gniazdo należy
sensor_name	nazwa czujnika zdefiniowana przez użytkownika
sensor_type	typ sensora
sensor_data	przechowywane dane z czujnika
sensor_data_date	data przechowywanych danych
sensor_data_time	czas przechowywanych danych
sensor_state	stan czujnika (włączone lub wyłączone)

Tabela 4.5 Opis atrybutów tabeli *sensors*

- Encja *socket_logs* służy do rejestracji przełączania gniazd. Umożliwia to, w przypadku, gdy system zachowuje się w sposób nieoczekiwany, na proste śledzenie, co i kiedy gniazdo włączyło bądź wyłączyło. Wymaga ona kilku kluczy obcych⁴ do poprawnego działania relacji: *adapter_id* oraz *socket_id*. Pierwszy wymieniony klucz oznacza, że ten konkretny log dotyczy listwy o identyfikatorze tutaj wskazanym i poprzez ten identyfikator mamy dostęp do wszystkich informacji z tabeli *adapters* na jego temat. To samo dotyczy *socket_id*. Pozwala to użyć np. na stronie internetowej dowolnych danych potrzebnych do opisu zapisu z rejestru zdarzeń gniazd. Opis encji znajduje się w tabeli 4.6.

Nazwa atrybutu	Zadanie
log_id	unikalny identyfikator zapisu, klucz główny
adapter_id	identyfikator listwy, do której gniazdo należy, klucz obcy
socket_id	identyfikator gniazda, klucz obcy
socket_state	zmiana na stan gniazda (włączenie lub wyłączenie)
last_changed	nazwa użytkownika
log_time	czas zmiany
log_date	data zmiany

Tabela 4.6 Opis atrybutów tabeli *socket_logs*

- Encja *socket_tasks* służy do przechowywania informacji o zadaniach automatycznych zdefiniowanych przez użytkownika systemu. Tabela ta będzie obsługiwana przez osobny podprogram o nazwie *taskManager*. Jej opis znajdziemy w tabeli 4.7.

⁴ Klucz obcy (foreign key) – jest to klucz, który definiuje relację poprzez powiązanie z kluczem podstawowym innej tabeli

Nazwa atrybutu	Zadanie
task_id	unikalny identyfikator zadania, klucz główny
socket_id	identyfikator gniazda, klucz obcy
sensor_id	identyfikator czujnika, klucz obcy
adapter_id	identyfikator listwy, do której gniazdo należy, klucz obcy
task_type	typ zadania
task_condition	warunek wykonania zadania
task_cycle	cykl powtarzalności zadania
task_state	stan zadania (wykonane lub w trakcie)
task_date	data zlecenia zadania
task_time	czas zlecenia zadania
task_user	nazwa użytkownika zlecającego zadanie
task_cancel	informacja dodatkowa np. o aktualizacji lub anulowaniu
task_active	stan zadania (wstrzymane lub aktywne)
task_error	błąd podczas przetwarzania zadania
task_time_controlled	sterowanie czasowe zadania
task_time_on	czas rozpoczęcia sterowania czasowego
task_time_off	czas zakończenia sterowania czasowego

Tabela 4.7 Opis atrybutów tabeli *socket_tasks*

Dla ścisłości, *task_state* informuje stronę internetową, czy zadanie zostało usunięte według zasady nic nie ginie. Jeśli tak się stanie, to zmieniamy jej stan na 0 i zadanie zostaje w systemie jako *wykonane*, a strona go nie wyświetla na liście. *Task_active* pozwala nam wstrzymywać pracę zadania bez konieczności usuwania i ponownego dodawania go do systemu.

- Encja *sensor_data* służy do przechowywania danych zbieranych przez podprogram *insertSensorData* z tabeli *sensors* w odstępach 15 minutowych w celu wizualizacji tych informacji na wykresach. Opis znajduje się w tabeli 4.8.

Nazwa atrybutu	Zadanie
data_id	unikalny identyfikator danych z czujnika, klucz główny
sensor_id	unikalny identyfikator czujnika, klucz obcy
data_date	data pobrania danych z czujnika
data_time	czas pobrania danych z czujnika
sensor_data	dane pobrane z czujnika

Tabela 4.8 Opis atrybutów tabeli *sensor_data*

- Encja *users* służy do przechowywania danych użytkowników oraz zaszyfrowanych algorytmem SHA-256⁵ haseł. Użytkownik nie może zostać usunięty, lecz co najwyżej zablokowany. Zapisywana jest, w celach bezpieczeństwa, informacja o ostatnim logowaniu użytkownika. Autor przewidział 3 rodzaje uprawnień dla użytkownika: *gość*, *użytkownik*, administrator. Jasnym jest, że administratorem urządzenia pozostaje autor projektu, gdyż tylko on będzie posiadał wiedzę o konfiguracji systemu oraz jego pracy. Więcej o prawach użytkowników piszemy w rozdziale 4.5. Opis encji przedstawia tabela 4.9.

Nazwa atrybutu	Zadanie
user_id	unikalny identyfikator użytkownika, klucz główny
user_login	nazwa użytkownika
user_password	zaszyfrowane hasło
user_permissions	uprawnienia użytkownika
user_blocked	flaga zablokowania użytkownika
user_last_login	informacja o czasie ostatniego logowania

Tabela 4.9 Opis atrybutów tabeli *users*

- Encja *job_list* służy do przechowywania zadań dla systemu nazywanych tutaj zleceniami, to znaczy zmian jego konfiguracji. Autor nie zna sposobu, aby bezpośrednio ze strony internetowej przesyłać informację do programu napisanego w C++ o chęci zmiany jego konfiguracji, dlatego wszelkie zmiany będą wprowadzane do tej tabeli, także w celu jej rejestracji i możliwości sprawdzenia czy polecenie zostało wykonane poprawnie lub jeśli nie, to dlaczego. Opisuje to tabela 4.10.

Nazwa atrybutu	Zadanie
job_id	unikalny identyfikator zlecenia, klucz główny
job_type	typ zlecenia
job_setting	opis ustawienia
job_setting2	opis ustawienia
job_setting3	opis ustawienia
job_active	stan zlecenia (włączone lub wyłączone)
job_date	data zlecenia
job_time	czas zlecenia
job_comment	informacja zwrotna z programu

Tabela 4.10 Opis atrybutów tabeli *job_list*

⁵ Algorytm SHA – zwany funkcją skrótu, jest to funkcja matematyczna pełniąca funkcję kryptograficzną, która kondensuje dane do żądanej długości, w tym przypadku 256 bitów/32 bajtów i polega głównie na hashowaniu.

- Encja *system_settings* służy do przechowywania podstawowych informacji systemu. Jest to konfiguracja, do którego będzie się odwoływać Raspberry i program zarządzania i będzie miał na nią wpływ użytkownik. Opis znajduje się w tabeli 4.11.

Nazwa atrybutu	Zadanie
setting_id	unikalny identyfikator ustawienia, klucz główny
setting_name	nazwa ustawienia
setting_value	wartość ustawienia

Tabela 4.11 Opis atrybutów tabeli *job_list*

- Encja *stats* służy do przechowywania informacji o wysłanych i odebranych pakietach. Opis znajduje się w tabeli 4.12.

Nazwa atrybutu	Zadanie
adapter_id	unikalny identyfikator adaptera, klucz główny
packet_recieved	liczba pakietów odebranych
packet_sent	liczba pakietów wysłanych
last_recieved_time	czas odebrania ostatniego pakietu
last_recieved_date	data odebrania ostatniego pakietu

Tabela 4.12 Opis atrybutów tabeli *stats*

Przedstawione w powyższej liście encje zostały wypracowane przy budowaniu programu oraz strony internetowej. Nie zmienia to faktu, że część z tabel, a w nich atrybutów, musiała zostać zaimplementowana na samym początku, aby program zarządzania mógł funkcjonować. Niektóre z atrybutów są nadmiarowe dla przyszłego zastosowania.

4.3. Oprogramowanie mikrokomputera Raspberry Pi

4.3.1. Protokół transmisji

Niestety moduł NRF24L01 nie pozwala na monitorowanie stanu połączenia, a wbudowany protokół z retransmisjami nie jest doskonały. Spowodowało to, że trzeba było wymyślić rozszerzenie istniejącego już protokołu. Rozszerzenie polega na tym, że każda listwa, aby nie nastąpiła kolizja wiadomości, ma dobrane odpowiednie opóźnienie zaczynające się od 10 ms w górę z krokiem 5 ms. Czas 5 ms przy retransmisjach trwających 4 ms jest dostateczny. Zapytania do listw wysyłane są jedno po drugim, aby dać adapterom czas na przetworzenie danych z czujników. Czasy przetwarzania i odbioru wiadomości mają pewien rozrzut czasowy w szczególności, jeśli włączymy tryb logowania na Arduino. Wynika to z tego, że Raspberry nadaje priorytety procesom, a Arduino dostają wiadomości w różnym czasie. Zostaje wyliczony czas maksymalnego oczekiwania na wiadomości według wzoru poniżej:

$$\text{Maksymalny_czas_oczekiwania_na_wiadomości} = \text{czas_wysłania_ostatniej_wiadomości} + \text{średni_czas_przetwarzania_i_odczytu_danych} + \text{opóźnienie} + \text{dodatkowa_zwłoka_czasowa}$$

Jeśli wiadomości dojdą w wyliczonym czasie, to zostaje im przypisane flaga *rxFlag*, która pozwala także ignorować wracające poprzez echo kopie wiadomości. Adapter zostaje dodany do listy do usunięcia z kolejki, by móc być od nowa dodany do niej w kolejnej pętli programu wraz z nowymi parametrami połączenia. W przypadku, gdy wiadomość nie dojdzie zostaje zwiększony licznik, przy wartości licznika np. 2 zostaje wprowadzony stan połączenia „słabe połączenie” a przy liczbie *waitNumber* stan „brak połączenia” i listwa zostaje wyłączona z obsługi z odpowiednią informacją oraz zostaje dodana do kolejki do usunięcia z listy zadań. Wartość licznik jest pobierana z encji *adapters* jako *adapter_weak_signal* dlatego, że każda listwa może mieć swój próg.

4.3.2. Program zarządzania siecią

Aplikacja, która tutaj zostanie opisana, ma bardzo ważne zadanie, jakim jest obsługa oraz wymiana informacji w sieci. Oprócz tego stawiane są jej inne mniejsze wymagania, takie jak: bieżąca kontrola stanu listw, zmiana konfiguracji sieci, kolejkovanie listw.

Opis kodu rozpoczniemy od głównej pętli programu, w której wywoływane są pewne kluczowe funkcje oraz inicjalizacja połączenia do bazy danych, a także wstępna konfiguracja nadajnika NRF24L01.

Kolejne kody będą dotyczyć metod, które wykonują określone zadania typu konwersja czy zapis do bazy, a co za tym idzie, są dużo mniej złożone.

Główna funkcja programu, która jest wywoływana jako pierwsza, nazywa się *main* i przyjmuje jako swoje parametry argumenty, poprzez które możemy wywołać program w różnych trybach pracy zdefiniowanych przez nas. Autor zdefiniował kilka takich trybów pracy, a są to:

- Tryb *logs*, który pozwala uruchomić program z trybem rejestrowania stanu systemu poprzez wyświetlanie informacji zdefiniowanych jako ważne z punktu widzenia administratora systemu.
- Tryb *logs button*, który pozwala na każdorazowe zatrzymanie wykonania pętli programu, która trwa około 300 ms (o tym dlaczego w kolejnym podrozdziale) oraz rejestrowanie stanu systemu.
- Tryb *logs slow*, który do trybu *logs* wprowadza jedynie opóźnienie pomiędzy pętlami w postaci 3 sekund.
- Tryb *logs rx*, który można wywołać w trybie *button* lub *slow* i pozwoli on pokazać w przypadku, gdy oczekiwana wiadomość zwrotna nie wróciła, jaka wiadomość w ogólności przysłała w czasie oczekiwania.

Do obsługi argumentów programu służy funkcja wywoływana na początku, *checkParameters*.

Tuż po tym następuje inicjalizacja ustawień nadajnika bezprzewodowego z wykorzystaniem biblioteki RF24. Tworzony jest obiekt klasy RF24 z parametrami jako numerami pinów na listwie wyprowadzeń GPIO (ang. General Purpose Input Output):

```
RF24 radio(5,1);
```

Rejestracja ustawień przebiega według listy poniżej:

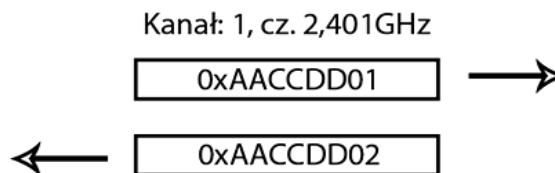
```
radio.begin();
```

```

radio.powerUp();
const uint64_t address_odbiornicy = 0xF0F0F0F0D2;
const uint64_t address_nadawczy = 0xF0F0F0F0E1;
radio.openReadingPipe(1,address_odbiornicy);
radio.openWritingPipe(address_nadawczy);
radio.setPALevel(RF24_PA_LOW);
radio.setDataRate(RF24_250KBPS);
radio.setCRCLength(RF24_CRC_16);
radio.setRetries(15,15);
radio.setChannel(25);

```

Urządzenie jest uruchamiane z tymi ustawieniami jako domyślnymi. Adresy odbiorcze i nadawcze różnią się na 2 ostatnich pozycjach. Pojęcie pipe'ów, które się tutaj pojawia jest ważne, gdyż oznacza kanał logiczny w kanale fizycznym. Moduł NRF24 pozwala na zestawienie 6 takich kanałów w trybie pół duplexowym, każde musi się różnić adresem. W programie jednak wykorzystane są tylko 2 do transmisji w obu kierunkach. Rysunek 4.5 przedstawia taki właśnie sposób transmisji.



Rys. 4.5 Pipe'y w kanale transmisyjnym

Moc nadawcza ustawiona jest na LOW, producent nie przedstawia konkretnych wartości prądowych. Prędkość transmisji, aby zapewnić dobrą jakość, ustawiona jest na najniższą, czyli 250 kb na sekundę. Do każdej ramki dostarczany jest cykliczny kod nadmiarowy CRC (ang. Cyclic Redundancy Check), który na podstawie algebry Boole'a wylicza 16 bitowy kod, który równa się sumie modulo (reszcie z dzielenia przez dwa) poszczególnych bitów, w celu zabezpieczenia transmisji i weryfikacji poprawnego odbioru. Dodatkowo domyślnie włączone są sprzętowe potwierdzenia ACK każdej wiadomości. Liczba retransmisji wynosi 15, a ich odstęp czasowy 15 x 250 us, czyli 4 ms [14]. Kanał transmisyjny został ustawiony jako ostatni, tj. 125. Leży on poza pasmem ISM (Industrial, Science, Medicine), które jest dostępne dla każdego. Jak wynika z noty katalogowej zakres częstotliwości pracy transmitera to 2400 MHz – 2525 MHz. Łatwo można policzyć, że 1 kanał ma szerokość pasma 1 MHz. Prawo mówi, że pasmo ISMs mieści się w przedziale 2.4 - 2.4835 GHz, także definiuje minimalne moce dla urządzeń [15]. Wykorzystane moduły mają jednak moc maksymalną 1 mW (0 dBm). Moc ta jest na tyle mała, że fala nie wychodzi na zewnątrz mieszkania bądź domu, oznacza to, że prawo nie będzie łamane.

Po stworzeniu obiektów z biblioteki *mysql++*:

```

MYSQL mysql;
MYSQL_ROW row;
MYSQL_ROW row2;
MYSQL_RES *result;
MYSQL_RES *result2;

```


można ich użyć do ustanowienia połączenia z bazą danych oraz odczytywania z niej danych. Kod poniżej przedstawia sposób, w jaki jest ustanawiane połączenie.

```
mysql_init(&mysql);
if (mysql_real_connect(&mysql, "127.0.0.1", "login", "hasło", "ipacs_database", 0, NULL, 0))
{
    while(1)
    {
        ...
    }
}
else
    printf("Błąd połączenia z bazą &mysql: %d, %s\n", mysql_errno(&mysql),
        mysql_error(&mysql));
```

Połączenie jest zestawiane z maszyną lokalną, dlatego też IP serwera wynosi 127.0.0.1. Reszta parametrów jest jasna lub z naszego punktu nieistotna.

W tym miejscu pojawia się problem opisany dalej w rozdziale 6.3.2 związany z uruchamianiem się usługi serwera MySQL.

Wracając do pętli głównej programu, mamy poza treściami logów spis głównych funkcji:

```
checkJobList();
taskQueueCheck();
taskExecute();
waitForAnswers();
checkRxFlags();
removeFromQueue();
```

Wykonywane one są bezparametrycznie, gdyż same w sobie wykonują zadanie w całości.

Poniżej przedstawione zostaną krótkie opisy tych funkcji:

- Metoda *checkJobList* jak sama nazwa wskazuje, sprawdza wcześniej opisaną tabelę *job_list* pod kątem nowo pojawiających się zleceń dla zmian konfiguracji systemu. Do takich zmian należą:
 - *powerLevelChange*, czyli zmiana mocy listwy. Przesłana zostaje wiadomość do adaptera i następuje oczekiwanie na wiadomość ACK.
 - *waitNumberChange*, czyli zmiana w tabeli *adapters* liczby prób połączenia z daną listwą.
 - *motherPoweLevel*, czyli zmiana mocy stacji matki, która jest możliwa po wyłączeniu nadajnika podczas wykonywania pętli programu.
 - *timeChange*, czyli zmiana czasu systemu w przypadku, gdy nie ma dostępu do Internetu.
 - *dateChange*, czyli zmiana daty systemu w przypadku, gdy nie ma dostępu do Internetu.
 - *motherRestart*, czyli restart systemu operacyjnego Raspian.
- Metoda *taskQueueCheck* jak nazwa wskazuje, polega na sprawdzeniu kolejki zadań, dodaniu do niej obsługiwanych listw i wybraniu parametrów transmisji według protokołu. Zadania w tym przypadku oznaczają połączenia (nadanie i odbiór). Funkcja działa w dwóch trybach:

- Kolejka jest pusta, następuje dodanie adapterów i wybranie dla nich parametrów transmisji.
- Kolejka zawiera adaptery, następuje poszukiwanie adaptera, który nie widnieje w kolejce i może być dodany oraz zostaje na koniec wyznaczone nowe opóźnienie transmisji dla każdego z nich.

W przypadku, gdy nie ma listw do dodania pętla programu zostaje opóźniona o 3 sekundy w celu zaoszczędzenia zasobów.

- Metoda *taskExecute* jest to część nadawcza programu. Tutaj zostają poprzez funkcje *prepareMessageToSend* oraz *getSocketStates* przygotowane wiadomości do wysłania do listwy.
- Metoda *waitForAnswers* jest częścią odbiorczą programu. Tutaj jest liczony maksymalny czas oczekiwania na odpowiedzi. Na początku wprowadzane jest stałe opóźnienie, gdyż wiadome jest, że wiadomości nie przyjdą wcześniej niż po 300 ms. W przypadku odebrania jakiegokolwiek wiadomości jest sprawdzana jej składnia. Dzieje się tak dlatego, że jak się okazało wiadomości mimo CRC16 potrafiły dojść z błędami. Prawdopodobnie w wyniku błędów przy transmisji w interfejsie SPI pomiędzy RPi a odbiornikiem. Dekodowany jest adres listwy i jeśli wiadomość jest echem to jest ignorowana, w przeciwnym razie zostaje przetwarzana dalej. Sprawdzany jest tutaj licznik prób połączenia i odpowiednio aktualizowany w bazie danych, aby użytkownik mógł zostać poinformowany o jego stanie. Wypakowywane zostają dane z czujników i poprzez funkcję *addDataToDB* dodane do bazy. Następnie następuje ustawienie flagi odebrania oraz sprawdzenie kontroli gniazd. Z powodów oszczędności zasobów dodane także zostało opóźnienie 1ms.
- Metoda *checkRxFlags* służy do weryfikacji flag *rx* po wykonaniu funkcji poprzedniej. W przypadku, gdy flaga ta jest w stanie *true* zadanie jest dodawane do kolejki do usunięcia z listy zadań. W przeciwnym razie sprawdzany jest stan licznika prób połączenia i jeśli któryś z zadań obecne na liście ma flagę *rx* w stanie *false*, to sprawdzany jest licznik prób i stan połączenia przyjmuje określoną wartość wcześniej już opisaną. Gdyby listwa osiągnęła wartość maksymalną prób i wiadomość by nie doszła, to w tym miejscu wyłączana jest jej obsługa. Funkcja ta dodatkowo udostępnia wyniki odebrania i wysłania pakietów, w celu generowania statystyki.
- Metoda *removeFromQueue* zarządza listą *removeList*, która zawiera listę adapterów, które muszą być usunięte z kolejki zadań w tej samej pętli programu. Jeśli lista nie jest pusta to adapter zostaje usunięty z *taskQueue* i odpowiednio zostaje ustawiony stan połączenia. Na koniec lista zostaje wyczyszczona.

Inne funkcje potrzebne do poprawnego działania programu to:

- *checkParameters* służy do weryfikacji argumentów uruchamiania programu,
- *getSocketStates* służy do pobrania z bazy danych informacji o gniazdach dla konkretnej listwy, które ID podajemy jako argument,
- *addDataToDB* służy do dodawania danych z czujników odebranych drogą radiową do tabeli *sensors*. Jako argument przyjmuje m.i. id adaptera oraz typ czujnika, aby można było odpowiedni sensor poprzez relację odnaleźć.

- *checkSensorState* służy sprawdzania czy określony czujnik jest włączony, aby nie przetwarzać z niego danych, jeśli w systemie został on zablokowany, np. z powodu wadliwego działania.
- *checkSensor* służy do sprawdzania id czujnika w danej listwie.
- *prepareMessageToSend* służy do przygotowania wiadomości do przesłania, to znaczy do stworzenia wiadomości o określonej strukturze: `0_AA_SSSS_DD`, gdzie pierwsza część to typ wiadomości kodowany jako cyfra (0 – żądanie, 1 – odpowiedź itd.), później dwucyfrowy adres adaptera, następnie 4 bitowy stan gniazd i opóźnienie, także dwucyfrowe.
- *taskQueueSend* służy do wysyłania wiadomości poprzez interfejs SPI.
- *checkExist* służy do sprawdzania czy dana listwa istnieje w kolejce zadań.
- *resetTxDelays* służy do resetowania przypisanych listwom opóźnień transmisji.
- *findTask* służy do wyszukiwania zadania w *taskQueue* dla listwy, której adres został odebrany w odpowiedzi na żądanie.
- *checkRecieved* służy do prewencji echa, sprawdza na podstawie flag *rx* czy wiadomość już wcześniej przyszła.
- *checkSockets* służy do sprawdzania czy w listwie są zdefiniowane 4 gniazda, aby można było poprawnie przetransportować wiadomość.
- *setControl* służy do prezentowania typu kontroli nad gniazdami. Użytkownik systemu może wymusić włączenie wszystkich gniazd poprzez przełącznik na listwie, a odpowiednia informacja pokaże się na stronie internetowej.
- *checkWaitNumber* służy do sprawdzania w bazie danych maksymalnej zdefiniowanej przez administratora liczby prób połączenia dla danej listwy.
- *checkWeakSignalIndicatorLevel* służy do sprawdzania w bazie danych progu słabego sygnału zdefiniowanego przez administratora dla danej listwy.

W programie wykorzystywane są pewne stałe, aby można było testować program ze zmienionymi parametrami pracy:

```
const int      startDelay = 10;
const int      delayDiff = 5;
const int      noAdapterDelay = 3000;
const int      answerDelay = 280;
const int      maxWaitTimeFactor = 360;
char  socketStates[4];
char  messageToSend[32]{};
```

Opis tych stałych zamieszczono poniżej:

- *startDelay* służy do dobrania początkowego opóźnienia dla nadania wiadomości zwrotnej po konwersji, musi to być wartość dwucyfrowa, aby zajęte były zawsze dwa znaki w ramce, stąd 10 ms,
- *deleyDiff* służy do dodania każdemu adapterowi tej wartości do opóźnienia transmisji,
- *noAdapterDelay* służy do wprowadzania opóźnienia w przypadku, gdy nie ma adapterów w kolejce, aby zaoszczędzić zasoby mikrokomputera,
- *answerDelay* służy do dodania wstępnego opóźnienia w odbiorze wiadomości w celu zaoszczędzenia zasobów,

- *maxWaitTimeFactor* jest to stała, którą dodaje się do *Maksymalny_czas_oczekiwania_na_wiadomości*, stała ta dobrana jest empirycznie,
- *socketStates* jest to tablica globalna, służąca do przechowywania stanu gniazd z ostatniej operacji sprawdzania,
- *messageToSend* jest to tablica globalna, służąca do przechowywania ramki do wysłania w formie znaków *char*. Funkcja *radio.write* wymaga jako parametru obiektu typu *char*, a nie adresu do obiektu. Wynika to z tego, że biblioteka *RF24* jest biblioteką pisaną w języku C i jest ona uniwersalną, wieloplatformową biblioteką. Dodatkowo NRF24L01 może jako dane przesyłać maksymalnie 32 bajty, co zostało tutaj wykorzystane.

Należy dodać, że kolejka zadań *taskQueue* to tablica typu *vector* obiektów klasy *Task*. Kolejka *removeList* natomiast jest to także tablica typu *vector* z tym, że obiektów typu *integer*, dlatego, że przechowywane są w niej tylko adresy listw.

Klasa *Task* została stworzona dlatego, że w pojedynczym zadaniu przechowywane jest kilka parametrów i odwoływanie się do nich poprzez indeksy w tablicy jest bardzo niewygodne, co więcej nieeleganckie od strony programistycznej. Można by było użyć wyliczeniowego typu danych *enum*, ale w celach ćwiczeniowych decyzja zapadła na korzyść obiektowości.

W klasie zapisane są prywatne stałe jako parametry oraz publiczne metody, za pomocą których możemy odczytać ze stałych ich wartości bądź je zapisać:

```
unsigned int adapterID;
unsigned int waitNumberCounter;
unsigned int maxWaitNumber;
unsigned int weakSignalIndicator;
unsigned int txDelay;
unsigned int sendingTime;
bool rxFlag;
public:
void setAdapterID(unsigned int);
unsigned int readAdapterID();
```

Funkcje wyglądają analogicznie dla każdej stałej, dlatego zostały przedstawione tylko *setAdapterID* oraz *readAdapterID*. Jeśli chodzi o stałe, to ich nazwy pojawiły się już wcześniej lub nazywają się na tyle jasno, że nie będziemy ich opisywać. Stała *sendingTime* przechowuje czas wysłania wiadomości, w celu wyliczenia maksymalnego czasu oczekiwania na wiadomość.

4.3.3. Program do wizualizacji

Program do wizualizacji służy do przepisywania danych z tabeli *sensors* do *sensorData* z odstępem 15 minutowym, aby można było na stronie graficznie przedstawić wyniki pomiarów.

Składa się on z 3 części, programu głównego *insertSensorData*, programu wywołującego program główny w odstępie 15 minut *insertSensorData_timeIntervalExecute* oraz pliku ze zdefiniowanymi na sztywno godzinami pomiarów *timeArray*, który wygląda następująco:

```

string timeArray[96] = {
    "00:00", "00:15", "00:30", "00:45",
    "01:00", "01:15", "01:30", "01:45",
    "02:00", "02:15", "02:30", "02:45",
    "03:00", "03:15", "03:30", "03:45",
    "04:00", "04:15", "04:30", "04:45",
    "05:00", "05:15", "05:30", "05:45",
    "06:00", "06:15", "06:30", "06:45",
    "07:00", "07:15", "07:30", "07:45",
    "08:00", "08:15", "08:30", "08:45",
    "09:00", "09:15", "09:30", "09:45",
    "10:00", "10:15", "10:30", "10:45",
    "11:00", "11:15", "11:30", "11:45",
    "12:00", "12:15", "12:30", "12:45",
    "13:00", "13:15", "13:30", "13:45",
    "14:00", "14:15", "14:30", "14:45",
    "15:00", "15:15", "15:30", "15:45",
    "16:00", "16:15", "16:30", "16:45",
    "17:00", "17:15", "17:30", "17:45",
    "18:00", "18:15", "18:30", "18:45",
    "19:00", "19:15", "19:30", "19:45",
    "20:00", "20:15", "20:30", "20:45",
    "21:00", "21:15", "21:30", "21:45",
    "22:00", "22:15", "22:30", "22:45",
    "23:00", "23:15", "23:30", "23:45"
};

```

Program do wywoływania dokładnie co 60000 ms, czyli co 1 minutę sprawdza, czy aktualny czas systemu to jeden z czasów tablicy *timeArray*, jeśli tak, to uruchamia program główny, jeśli nie, to ponownie jest usypiany na 60 sekund.

Program główny jest nieskomplikowany dlatego, że wykonuje on proste operacje na tablicach w bazie MariaDB. Operacje wykonywane to:

- 1) połączenie się z bazą *ipacs_database*,
- 2) pobranie adresów adapterów, które są obsługiwane,
- 3) pobranie danych z włączonych sensorów z encji *sensors*,
- 4) wstawienie pobranych danych do tablicy *sensorData* wraz z aktualnym czasem.

Okres, z jakiego będzie można zwizualizować dane został wybrany 15 dni, aby nie był zbyt „poszarpany” i gęsty.

4.3.4. Program menadżera zadań

Program ten uruchamia się jako osobny proces i działa niezależnie od pozostałych dwóch. Zadania w rozumieniu menadżera zadań to zlecenia od użytkowników. Po utworzeniu połączenia z bazą danych w pętli głównej programu wywoływane są co 100 milisekund 2 funkcje:

- *checkTasks* skanuje tablicę *socket_tasks* w celu poszukiwania zlecenia, które jest aktywne i niewykonane. Najpierw jest sprawdzany stan sensora oraz czy nie ma błędu wcześniej zapisanego w zleceniu. Następnie sprawdzane są typy zadań oraz warunki dla konkretnego zlecenia. Dla zadań kontrolowanych czasowo, czyli tylko w danym przedziale czasowym, jest on pobierany i przeliczany, aby można było go porównać z aktualnym. Jeśli czas dla zadania zostanie przekroczony to gniazdo zostaje wyłączone, ale zadanie pozostaje dalej aktywne. W przypadku, gdy warunek podany przez użytkownika nie zgadza się z założonym w systemie, zostaje zapisany i wyświetlony na stronie błąd zadania. Zostaje ono automatycznie zatrzymane. Jeśli warunki są spełnione, to zadania zostają zapisane w tablicy typu *vector* z obiektami typu *long* przechowującymi identyfikatory zadania.
- *executeTasks* zmienia stan gniazda w zależności od typu zlecenia. Jeśli jest ono typu *on* to sprawdza aktualny stan i jeśli jest *off*, to zmienia stan na 1, w przeciwnym razie nie robi nic. Jeśli zlecenie w takim wypadku jest powtarzalne, to zostaje aktywne oraz zostaje przesłana informacja o czasie aktualizacji zadania. Jeśli zadanie nie było powtarzalne to zostaje zapisana informacja o czasie zakończenia. Następnie zostaje wyczyszczona lista zadań z kolejki *tasksToExecute*.

4.4.Oprogramowanie platformy Arduino Nano

Program pisany dla Arduino, jest względnie prosty. Język w jakim jest on pisany, jest bardzo łatwy i zrozumiały.

Wcześniej wspomniano o tym, że biblioteka *RF24* jest pisana na różne platformy. W tym przypadku, w większości współgra ona ze środowiskiem Arduino IDE. Dlatego też sama inicjalizacja odbiornika NRF jest identyczna jak w RPi z tą różnicą, że adres odbiorczy teraz jest adresem nadawczym i odwrotnie.

Dane z czujnika temperatury, który jest sensorem inteligentnym, są przesyłane poprzez protokół *OneWire*, czyli jednym drutem. Po dostarczeniu biblioteki *OneWire* oraz *DallasTemperature* możliwe jest łatwe otrzymywanie z niego danych. Precyzja została ustawiona na 10 bitów, co daje czas konwersji na poziomie 188 ms oraz możliwość pomiaru do ćwierci stopnia Celsjusza.

Program po inicjalizacji w pętli głównej rozpoczyna od odbioru. Jeśli moduł odbierze wiadomość, zostaje ona zdekodowana. Jeśli ma ona typ 4, to oznacza, że jest to wiadomość administracyjna o zmianie mocy adaptera. Zostaje to wykryte na początku wykonywania programu. Nadajnik zostaje przestawiony i wysyłana jest informacja zwrotna. Należy tutaj zaznaczyć, że wiadomość typu 4 w RPi jest wysyłana 20 razy, w przypadku, gdy nie wróci wiadomość zwrotna potwierdzająca zmianę mocy. Jeśli wiadomość przychodząca jest typu 0 i została zaadresowana do danego nadajnika, to następuje ustawienie wyjść gniazd w odpowiednie dla nich stany. Sprawdzany jest stan przełącznika ręcznego sterowania, jeśli jest on w stanie 0 to wszystkie gniazda zostają mimo wszystko załączone. W tym momencie zostaje wprowadzone opóźnienie transmisji i następuje część nadawcza. Odczytywana jest z przetwornika analogowo-cyfrowego wartość natężenia światła, po czym jest on konwertowany do skali procentowej. Następnie poprzez funkcję *requestRemperatures* na obiekcie *DallasTempepratures* następuje odczyt z rejestrów czujnika *DS18B20*. Kolejny etap

to utworzenie wiadomości nadawczej wraz z danymi już przetworzonymi, gotowymi do transmisji oraz z informacją o sterowalności gniazd.

Aby można było zdalnie podejrzeć stan listwy zostały dodane logi do programu. Poprzez moduł Bluetooth, który wykorzystuje protokół szeregowy do transmisji danych wyjściowych można bez przeszkód odczytywać stan platformy przez telefon komórkowy.

4.5.Oprogramowanie strony internetowej

Oprogramowanie strony internetowej to plik ze „stylem” strony, czyli opisem w języku CSS wyglądu sekcji i elementów strony oraz pliki szkieletowe wraz z skryptami do jej obsługi. Autor podszedł do zadania według zasady „dziel i zwyciężaj”, czyli jeśli można coś uprościć i rozbić, to tak robimy. Każda podstrona ma swój osobny plik, większość skryptów wprowadzających modyfikację w bazie danych ma także osobny plik.

Opis nie będzie się skupiał na kodzie, a na funkcjach zaimplementowanych i sposobie ich działania, tak aby funkcjonowanie strony było zrozumiałe i jasne. Pod uwagę trzeba wziąć dużą ilość plików, które razem tworzą całość. Dlatego, aby opis był spójny, każdy z większych elementów strony zostanie przedstawiony w osobnym podrozdziale.

4.5.1. Logowanie, uprawnienia

Z powodów bezpieczeństwa strona jako jedno z zabezpieczeń została wzbogacona o uprawnienia. Każdy użytkownik posiada jasno zdefiniowane uprawnienia, a ich opis przedstawia tabela 4.12.

Bezpieczeństwo należy rozumieć poprzez działania osób trzecich w celach manipulacji systemem według własnych, często złych pobudek oraz poprzez ochronę systemu przed przypadkową zmianą konfiguracji przez jego użytkownika. Uprawnienia nadane użytkownikowi nie spowodują katastrofalnych problemów w działaniu sieci.

Konto użytkownika zostaje zabezpieczone poprzez kodowanie hasła metodą SHA-256. Wykonywane jest to w pliku, który jest wywoływany po kliknięciu „Zaloguj się”. W tym momencie użytkownik zostaje uwierzytelniony, czyli następuje porównanie wpisanego loginu oraz hasła, z tymi w bazie danych. Jeśli operacja się zakończy sukcesem to użytkownik zostanie zalogowany, co oznacza, że zostanie zestawiona sesja⁶ na bazie ciasteczek⁷. W przeciwnym wypadku zostanie wyświetlona informacja zwrotna, rys. 4.6.

⁶ Sesja (ang. session) – mechanizm pozwalający na przesyłanie informacji pomiędzy stronami. Pozwala on przechowywać dane w ciasteczkach bądź w adresie URL. U użytkownika przechowywany jest tylko ID sesji, a reszta po stronie serwera [16].

⁷ Ciasteczka (ang. cookies) – pliki przechowywane na urządzeniu użytkownika, które przechowują dane informatyczne, w celach optymalizacji lub poprawnego funkcjonowania skryptów. Mogą być sesyjne (tymczasowe) lub stałe [17].

Typ użytkownika	Dostępne uprawnienia
gość	<ul style="list-style-type: none"> Możliwość przeglądania strony Możliwość podglądu stanu systemu w panelu konfiguracyjnym Brak możliwości zmiany jakiegokolwiek elementu w systemie Brak możliwości definiowania zadań automatycznych Brak możliwości podglądu zadań automatycznych listwy Możliwość zmiany hasła oraz czasu odświeżania
użytkownik	<ul style="list-style-type: none"> Możliwość przeglądania strony Możliwość podglądu stanu systemu w panelu konfiguracyjnym Możliwość definiowania, usuwania, wstrzymywania zadań automatycznych Możliwość przełączania gniazd Możliwość przywrócenia połączenia Możliwość zmiany nazwy listwy oraz gniazd Możliwość resetowania połączenia i włączania obsługi listw Możliwość zmiany hasła oraz czasu odświeżania Możliwość zmiany daty i czasu systemu Możliwość zmiany mocy stacji matki
administrator	<p>Posiada uprawnienia użytkownika oraz dodatkowo:</p> <ul style="list-style-type: none"> Możliwość dodawania i dezaktywacji listwy Możliwość zmiany mocy, liczby prób połączenia i progu słabego połączenia listwy Możliwość dodawania, wyłączania, zmiany nazwy i usuwania sensora listwy Możliwość dodawania, usuwania gniazd Możliwość podglądu logów zmian konfiguracji oraz zmian wszystkich gniazd Możliwość restartu systemu z poziomu strony internetowej Możliwość dodawania użytkowników Możliwość zmiany hasła i uprawnień użytkowników Możliwość blokowania i odblokowywania użytkowników

Tabela 4.12 Uprawnienia użytkowników

Sesja zawiera informacje o identyfikatorze użytkownika, jego uprawnieniach oraz nazwie. W tym przypadku użyta jest także zmienna *error* do przesłania informacji o błędzie logowania. Jeśli logowanie się zakończyło, to zostaje ustawiona flaga *logged*.

W przypadku, gdy użytkownik próbuje bez logowania lub uprawnień przejść na konkretną podstronę zostaje to wykryte i zablokowane. Dzieje się tak dlatego, że w pierwszej kolejności na każdej podstronie sprawdzane są warunki sesji, czyli czy flaga *logged* jest ustawiona w danej sesji, jeśli tak się nie stało, to strona przekieruje automatycznie użytkownika do strony logowania.

Praca dyplomowa magisterska pt.

SYSTEM INTELIGENTNEGO STEROWANIA LISTWAMI ZASILANIA

Tematem pracy jest System Inteligentnego Sterowania Listwami Zasilania. Określenie inteligentny należy rozumieć tutaj jako automatyzację działania systemu, a więc ograniczenia interakcji systemu z użytkownikiem do minimum po pełnej automatyzacji procesu. Projekt polega na opracowaniu systemu modułów dołączanych do listw zasilania przedłużaczy 230V oraz zapewnieniu jego działania od strony programowej. W każdym z gniazd użyte będą czujniki, aby można było gniazda sterować za ich pomocą. Sterowanie odbywać się będzie poprzez stronę WWW w wersji komputerowej i mobilnej, na smartfonie, z zabezpieczoną domową siecią lokalną oraz panelem logowania z uprawnieniami. Na stronie będzie można załączać i wyłączać gniazda, ustawiać automatyczne zadania oraz sprawdzić zapisy z dziennika zmian stanów. Dostępny będzie panel konfiguracyjny do wprowadzania zmian w systemie oraz szczegółowe dane z czujników w postaci czasowych wykresów z wyborem przedziału czasu.

Celem projektu jest wprowadzenie do domu możliwości zdalnego zarządzania wybranymi urządzeniami poprzez ich automatyzację lub ręczną kontrolę. System pozwoli także na kontrolę temperatury w okolicach listw zasilających oraz sprawdzenie, jak zmieniło się natężenie światła w ciągu np. ostatniej doby. Zadania automatyczne mają zwolnić użytkownika z włączania oświetlenia w momencie, gdy w pomieszczeniu bądź na zewnątrz robi się lub uruchamiania grzejnika elektrycznego, gdy temperatura spadnie poniżej zadanego poziomu.

Login:
login

Hasło:
hasło

Zaloguj się

WERSJA
MOBILNA
STRONY

Rys. 4.6 Panel logowania

4.5.2. Panel główny

Zadaniem panelu głównego, zwanego stroną główną w języku stron internetowych, jest przedstawienie użytkownikowi elementów kluczowych systemu.

Kluczowe elementy są tutaj rozumiane poprzez:

- Menu główne, które wyświetla odnośniki do podstron dotyczące całego systemu, a nie poszczególnych listw. Wygląd tego menu został przedstawiony na rysunku 4.7.

Praca dyplomowa magisterska pt.

SYSTEM INTELIGENTNEGO STEROWANIA LISTWAMI ZASILANIA

Informacje Podgląd zadań automatycznych Panel konfiguracyjny Zalogowano jako wojtek

Rys. 4.7 Menu główne

- Spis listw dostępnych w sieci, które opiera się na stworzeniu odseparowanych od siebie kontenerów, w których są bloki z pozostałymi danymi dotyczącymi adaptera. Taką przestrzeń obrazuje rysunek 4.8.
- W każdym module jest spis gniazd, który przy kolorze czerwonym sygnalizuje wyłączone gniazdo, przy zielonym gniazdo załączone. W przypadku, gdy gniazdo jest sterowane ręcznie następuje jego blokada. Tak samo się dzieje w przypadku, gdy mamy gniazdo sterowane przez zadanie automatyczne i nie jest ono odblokowane (domyślnie tak nie jest). Po najechnięciu kursorem na przycisk zostanie przedstawiona także odpowiednia informacja (rys. 4.9).

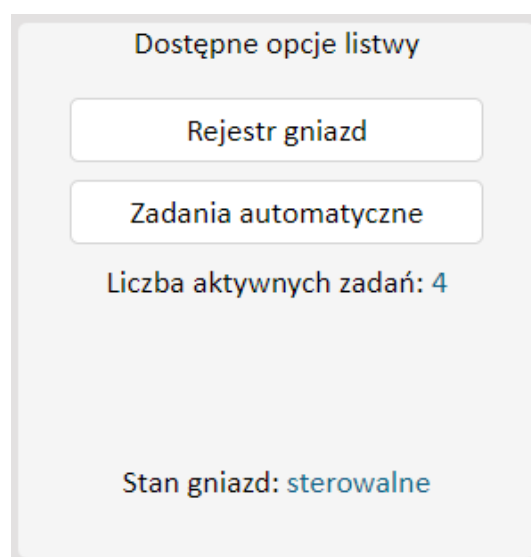


Rys. 4.8 Interfejs blokowy

- Dostęp do rejestru oraz zadań automatycznych w każdym adapterze. W panelu dodatkowo mamy informacje o liczbie aktywnych zadań dla danej listwy oraz czy jest ona sterowalna ręcznie (stan *zablokowane*) bądź poprzez stronę (stan *sterowalne*, rys. 4.10).



Rys. 4.9 Panel gniazd



Rys. 4.10 Opcje listwy

- Dostęp do danych i wykresów z czujników w każdej listwie. Dane te są prezentowane w postaci rzeczywistej oraz z okresu 24-godzinnego. Jeśli dany sensor jest wyłączony, zostaje w jego miejscu przedstawiona taka informacja, a jeśli temperatura przekroczy 30 stopni Celsjusza, to zmieni ona swój kolor na czerwony. Blok został zaprezentowany na rysunku 4.11.

W następnych podrozdziałach zajmiemy się opisem wspomnianych tutaj elementów systemu w sposób bardziej szczegółowy.

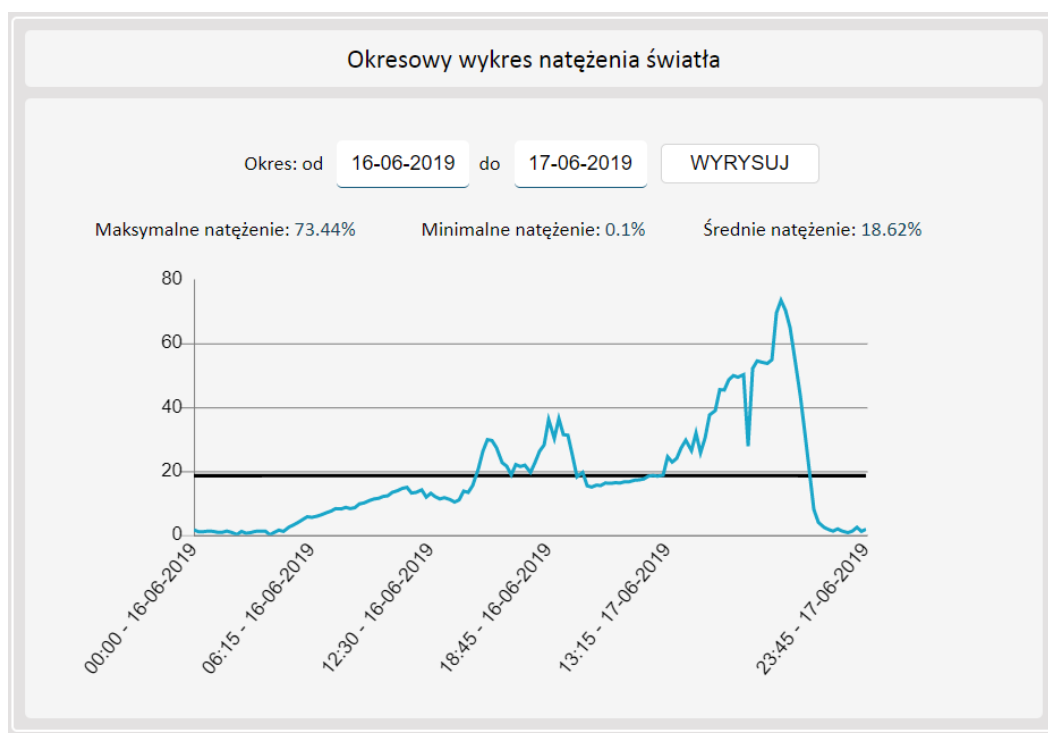


Rys. 4.11 Opcje listwy

4.5.3. Panel wizualizacji

Panel wizualizacji służy do prezentowania wyników pomiarów temperatury oraz natężenia światła w pobliżu listwy. Jest to funkcja dodatkowa i nie wpływa na działanie systemu. Użytkownik może przeanalizować zmiany wielkości mierzonych i na tej podstawie zautomatyzować pracę adaptera.

Określony został maksymalny przedział czasu jako 15 dni, aby wykres był możliwie czytelny. Po wejściu na podstronę zostaną wyświetlone grafy z ostatniej znanej daty zebrania danych. Tak samo się dzieje, jeśli użytkownik poda zły okres. Każdy rysunek generowany jest osobno, aby zmienić czas w drugim wykresie należy ponownie wybrać okres datowy. W przypadku, gdy czujnik jest wyłączony, informacja zostaje także przedstawiona w panelu. Dodatkowo została naniesiona czarna linia, która wykreśla średnią wartość. Wykres jednej wielkości mierzonej został przedstawiony na rysunku 4.12.



Rys. 4.12 Wykres natężenia światła

Wykresy są generowane w języku skryptowym *JavaScript* za pomocą biblioteki *ChartJS*. Długości u dołu tworzą automatyczne odstępy i kiedy ilość pomiarów i szerokość okna się nie zgadza, tworzony jest większy odstęp. Dlatego też jest widoczny na rysunku powyżej. Jest to jawny minus tego rozwiązania.

4.5.4. Panel zadań automatycznych

Panel zadań automatycznych jest dość zaawansowanym mechanizmem obsługiwany przez menadżer zadań. Pozwala na dodawanie zadań, a także ich podgląd. Dodanie zadania polega na podaniu jego typu, czasu pracy (jeśli taki chcemy), warunku, który spełnia ustaloną normę (w przeciwnym wypadku będzie wyświetlony błąd warunku i zadanie zostanie trwale wstrzymane), wyborze gniazda spośród gniazd listwy oraz wyborze typu powtarzalności. Zadania jednorazowe zostaną wykonane tylko raz, gdy warunek zostanie spełniony, zadania powtarzalne (wielorazowe) zostaną wykonane przy każdym spełnieniu warunku i jeśli wybierzmy kontrolę czasową, w danym przedziale czasowym. Wygląd panelu dodawania przedstawiony został na rysunku 4.13.

Rys. 4.13 Panel dodawania zadania

Panel zarządzania zadaniami pozwala podejrzeć zleczone zadania oraz nimi sterować w sposób ograniczony. Autor nie zdecydował się na możliwość edycji zadania, gdyż mogło to spowodować dalszą kompilację kodu, w przypadku gdyby użytkownik podał złe warunki po zmianie itd. Dodawanie kolejnych zadań jest na tyle szybkie i proste, że postawiono tylko na możliwość ich wstrzymywania (poprzez czerwoną/zieloną kropkę) i trwałego usuwania. Istnieje także możliwość wyłączenia blokady gniazda, która domyślnie się załącza, w celu poinformowania użytkowników, że gniazdo jest sterowane poprzez menadżer zadań. W przypadku wcześniej opisanego błędu warunku, także zostanie on tutaj wyświetlony. Istnieje możliwość wyczyszczenia go z panelu konfiguracyjnego. Dodatkowo, na dole poza spisem mamy opcje dotyczące całej listy zadań, czyli szybkiego wstrzymania wszystkich

zadań oraz ich usunięcia. W tabeli jest też dostępna ikona zegara, która informuje o tym, że zadanie jest kontrolowane czasowo i po najechaniu kursorem podawany jest przedział czasu. Takich zadań kontrolowanych czasowo można dodać nieograniczoną ilość, nawet mogą się one nachodzić. Niestety może to doprowadzić do tego, że gniazdo rozpocznie zmieniać stan z jednego na drugie i z powrotem. Panel ten przedstawia rysunek 4.14.

Menadżer zadań listwy									
ID	stan	gniazdo	typ zadania	warunek	powtarzalność	użytkownik	czas i data	blokada	błąd
242	●	Lampka nocna	Włącz o wskazanej godzinie	00:00	powtarzalna	Wojtek	23:40 04-06-2019		<input type="button" value="Usuń"/>
240	●	Lampka nocna	Wyłącz, gdy natężenie światła wzrośnie powyżej	10	powtarzalna	wojtek	21:04 28-05-2019		<input type="button" value="Usuń"/>
238	●	Lampka nocna	Włącz, gdy natężenie światła spadnie poniżej	9	powtarzalna	wojtek	21:03 28-05-2019		<input type="button" value="Usuń"/>
237	●	Pasek LED	Wyłącz, gdy natężenie światła wzrośnie powyżej	10	powtarzalna	wojtek	20:33 27-05-2019		<input type="button" value="Usuń"/>
236	●	Pasek LED	Włącz, gdy natężenie światła spadnie poniżej	9	powtarzalna	wojtek	20:33 27-05-2019		<input type="button" value="Usuń"/>

Początek: 20:00
Koniec: 00:00

Gniazda po zdefiniowaniu zadania są automatycznie blokowane, aby je odblokować należy nadusić otwartą kłódkę.
Aby zatrzymać wykonanie zadania należy nacisnąć zieloną/czerwoną kropkę przy kolumnie stan.
Zatrzymaj aktywne zadania Usuń wszystkie zadania

Rys. 4.14 Panel podglądu zadań

4.5.5. Panel dziennika gniazd

Panel dziennika gniazd jest prostym narzędziem do weryfikowania stanu gniazd. Jeśli nie zachowują się one tak jak powinny lub chcemy sprawdzić, kto i o której godzinie wyłączył dany slot i w przejrzysty sposób się tutaj tego dowiemy. Wyświetlane jest 30 ostatnich zmian w postaci tabeli. Administrator dodatkowo ma możliwość podejrzenia zmian na wszystkich listwach jednocześnie, ale opcja ta jest dostępna dla niego z poziomu panelu konfiguracyjnego. Jeśli stan gniazda zostanie zmieniony przez zadanie automatyczne, to informacja ta zostanie zawarta w miejscu „kto”, jako *Menadżer zadań*. Panel rejestru przedstawia rysunek 4.15.

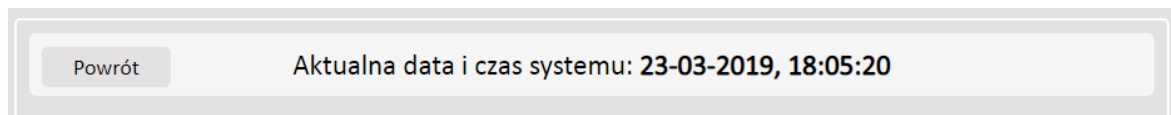
Dziennik zmian stanów gniazd						
<input type="button" value="Powrót"/>						
ID logu	adapter	gniazdo	stan gniazda	kto	czas	data
1600	614, za łóżkiem	Lampka	włączenie	Menadżer zadań	17:38	23-03-2019
1597	614, za łóżkiem	Lampka	wyłączenie	wojtek	10:31	18-03-2019
1596	614, za łóżkiem	Lampka	włączenie	wojtek	10:31	18-03-2019
1595	614, za łóżkiem	Kino domowe	wyłączenie	admin	23:36	10-03-2019
1594	614, za łóżkiem	Kino domowe	włączenie	admin	23:36	10-03-2019
1593	614, za łóżkiem	Dekoder	wyłączenie	admin	21:48	10-03-2019
1592	614, za łóżkiem	Dekoder	włączenie	admin	21:48	10-03-2019
1591	614, za łóżkiem	Lampka	wyłączenie	wojtek	08:43	10-03-2019
1590	614, za łóżkiem	Lampka	włączenie	wojtek	08:43	10-03-2019

Rys. 4.15 Fragment panelu rejestru gniazd

4.5.6. Panel konfiguracyjny

Panel konfiguracyjny jest dość skomplikowanym elementem. Pozwala na rekonfigurację większości elementów systemu. Wykorzystywane są tutaj skrypty JS, dużo zapytań do bazy oraz komendy systemowe typu *bash*, ale w postaci zadań konfiguracji. Dużą rolę grają tutaj uprawnienia użytkownika. Analizę bloków będzie przeprowadzana na koncie administratora, z racji tego, że posiada on wszystkie niezbędne uprawnienia do wyświetlania opcji konfiguracji.

Po wejściu na podstronę ukazuje się nam schemat blokowy paneli, z których każdy ma inne zastosowanie. Pierwszym jest aktualny czas systemu, który jest istotny dla użytkowników systemu. W przypadku, gdy system nie jest zsynchronizowany z „serwerem czasu”, wtedy system będzie działał według innego czasu i spowoduje to nieprawidłowe jego działanie (rys. 4.16).



Rys. 4.16 Panel czasowy

Drugi z paneli to panel stanu systemu, można tutaj podejrzeć stan każdego gniazda, sensora oraz listwy. Każdy z tym elementów ma w kolumnach do dyspozycji ograniczone opcje, które widać przy każdym podzespole na rysunku 4.17.

Stan systemu		
adapter	czujniki	gniazda
<p>Listwa nieobsługiwana</p> <p>Status gniazd: sterowalne</p> <p>ID listwy: 1 Pokój: <i>Wojtka</i> Miejsce: <i>przy łóżku</i></p> <p>Stan połączenia: słabe Odpytywanie: wyłączone Statystyka: 0 Moc nadawcza listwy: MIN Próg słabego połączenia: 6 Liczba prób połączenia: 30</p> <p>Dostępne opcje dla listwy:</p> <ul style="list-style-type: none"> Włącz odpytywanie listwy Anuluj aktywne zadania Zatrzymaj aktywne zadania Wyczyść błędy menadżera Dezaktywuj listwę Zmień nazwę <p>Zarządzenie połączeniem:</p> <ul style="list-style-type: none"> Ustaw liczbę prób połączenia Ustaw próg słabego sygnału Zmień moc listwy 	<p>Czujnik temperatury włączony</p> <p>Sensor ID: 1 <i>DS18B20</i> Dane: 22°C Czas i data: 14:36:28, 06-01-2019</p> <ul style="list-style-type: none"> Wyłącz sensor Usuń sensor Zmień nazwę <p>Czujnik światła włączony</p> <p>Sensor ID: 6 <i>fotorezystor</i> Dane: 32% Czas i data: 14:36:28, 06-01-2019</p> <ul style="list-style-type: none"> Wyłącz sensor Usuń sensor Zmień nazwę 	<p>Socket ID: 1 <i>Gniazdo testowe 1</i> Gniazdo włączone Gniazdo sterowane ze strony</p> <ul style="list-style-type: none"> Usuń gniazdo Zmień nazwę <p>Socket ID: 2 <i>Blabla</i> Gniazdo wyłączone Gniazdo sterowane ze strony</p> <ul style="list-style-type: none"> Usuń gniazdo Zmień nazwę <p>Socket ID: 3 <i>Gniazdo testowe 3</i> Gniazdo włączone Gniazdo sterowane ze strony</p> <ul style="list-style-type: none"> Usuń gniazdo Zmień nazwę <p>Socket ID: 20 <i>Gniazdo testowe 4</i> Gniazdo włączone Gniazdo sterowane ze strony</p> <ul style="list-style-type: none"> Usuń gniazdo Zmień nazwę

Rys. 4.17 Panel stanu systemu

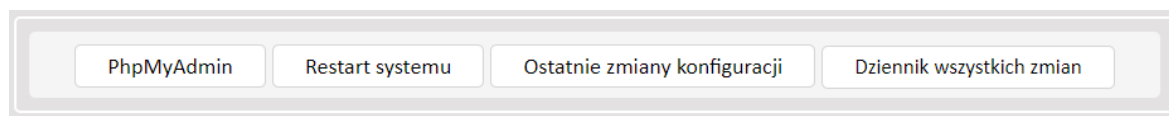
Rysunek 4.17 dodatkowo przedstawia sytuację, gdy gniazda są sterowane ręcznie, czyli listwa jest obsługiwana częściowo. Oznacza to, że jest brak możliwości zmiany stanów gniazd, są one ręcznie załączone. Co więcej, stan połączenia jest słaby. Informacja ta jest także wyświetlana, pod nazwą listwy. Pod częścią opcji po kliknięciu pojawią się dodatkowe komponenty (rys. 4.18). Treść pola wpisywania tekstu jest automatycznie wypełniana aktualną wartością.



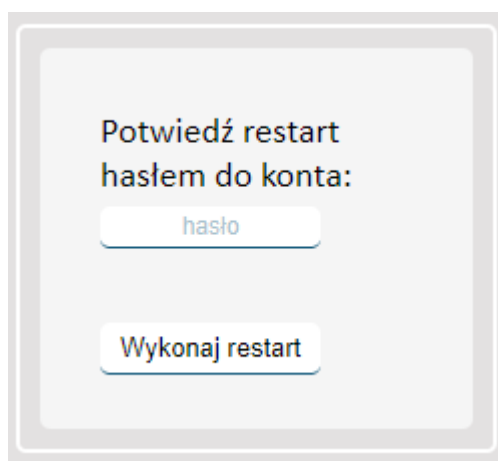
Rys. 4.18 Dodatkowe komponenty

Zmiana mocy listwy, zmiana progu słabego sygnału oraz liczba prób zostają zapisane na liście zadań konfiguracji i część z nich zostaje przetworzona przez program, bo wymaga przesłania lub rekonfiguracji samego programu.

Kolejny panel to panel menu (rys. 4.19) dostępnego tylko dla administratora. Pierwszym elementem jest odnośnik do panelu zarządzania bazą danych *phpMyAdmin*. Drugi człon służy do wykonania restartu bez konieczności logowania się zdalnego na serwer. Operację tę można wykonać poprzez podanie ponowne hasła do konta (rys. 4.20).



Rys. 4.19 Menu administracyjne



Rys. 4.20 Podstrona restartu systemu

Następnie mamy opcję „Ostatnie zmiany konfiguracji”, która pozwala przeglądać zlecone zmiany oraz sprawdzić czy zostały one wykonane (rys. 4.21). Ustawienie 3 zostało utworzone do przyszłych zastosowań.

Jako ostatni pojawia się wcześniej wspomniany rejestr zmian stanów gniazd (dla wszystkich listw) oraz opcja restartu systemu.

Powrót		Ostatnie zmiany konfiguracji						
ID zmiany	typ zmiany	ustawienie 1	ustawienie 2	ustawienie 3	czas	data	aktywność	komentarz
305	weakSignalLevelChange	2	2		14:16	17-03-2019	0	Job finished
304	weakSignalLevelChange	2	1		14:16	17-03-2019	0	Job finished
303	weakSignalLevelChange	2	5		14:14	17-03-2019	0	Job finished
302	weakSignalLevelChange	2	1		14:06	17-03-2019	0	Job finished
301	weakSignalLevelChange	2	0		14:02	17-03-2019	0	Job finished
300	weakSignalLevelChange	2	1		14:01	17-03-2019	0	Job finished
299	powerLevelChange	2	HIGH		14:00	17-03-2019	0	job finished

Rys. 4.21 Podstrona zmian konfiguracji

Kolejnym blokiem sterującym jest panel ustawień stacji bazowej. Jest on dostępny już dla użytkownika, znaczy to, że ma on możliwość ingerencji w proste elementy systemu. Zmiana mocy stacji matki nie wpłynie znacząco na pracę systemu. Wynika to z faktu, że w prosty sposób można przywrócić poprzednie ustawienie. Blok ten przedstawia rysunek 4.22.

Ustawienia stacji matki

Zmień godziny i daty

Podaj nową godzinę:

Podaj nową datę:

HH:mm:ss

RRRR-MM-DD

Zmień godzinę

Zmień datę

Zmień mocy nadajnika

Wybierz moc nadawczą:

Niska

Zmień moc

Rys. 4.22 Panel ustawień stacji bazowej

Przedostatni człon to panel zarządzania użytkownikami. Można tutaj między innymi dodać użytkownika, zmienić mu hasło bądź zmienić jego uprawnienia lub go zablokować. Po każdej operacji wyświetlane jest potwierdzenie zmian (rys. 4.23).

Ustawienia użytkowników

Dodawanie użytkownika

Podaj nazwę użytkownika:

login

Podaj hasło użytkownika:

hasło

Powtórz hasło:

hasło

Wybierz typ konta:

użytkownik

Dodaj użytkownika

Zmień hasło

Zmień uprawnień

Zmieniono uprawnienia / odblokowano użytkownika!

Wybierz użytkownika:

admin [admin, 11-03-2019, 09:12]

Wybierz typ konta:

użytkownik

Zmień uprawnienia i odblokuj

Zablokuj użytkownika

Rys. 4.23 Panel zarządzania użytkownikami

Oczywiście do omawianego panelu uprawnienia ma tylko administrator, natomiast do ostatniego mają już wszyscy, w tym gość. Jest nim panel ustawień konta (rys. 4.24). Można tutaj zmienić hasło do konta oraz czas automatycznego odświeżania strony. Wartość maksymalna to 9999, która jest na tyle duża, że można ją uznać za brak odświeżania. Domyślna wartość to jedna minuta. Odświeżana jest strona główna oraz panel konfiguracyjny.

Rys. 4.24 Panel ustawień konta

4.5.7. Wersja mobilna strony

W poprzednich podrozdziałach zostały przedstawione elementy strony internetowej w wersji „desktopowej”, czyli komputerowej. Wiadome jest, że każdy woli użyć smartphona niż podejść do komputera, dlatego duża część strony została także przeniesiona do wersji mobilnej. Taka reprezentacja interfejsu nie jest prosta, gdyż ekrany są zazwyczaj w proporcji 9:16, co oznacza, że są węższe, gdy na nie patrzymy. Dla porównania ekrany komputerów typowo są w proporcji 16:9. Dodatkowo widok powinien się skalować do wyświetlacza.

Po dodaniu metatagu⁸ do każdej podstrony udało się osiągnąć poprawną skalowalność interfejsu, jednakże należało dopasować treść pod kątem ekranów mobilnych.

Przedstawienie każdej podstrony i ponowny opis były powtórzeniem, bo zmienił się tylko niejako styl i wygląd strony. Dlatego też autor zdecydował pokazać wygląd strony jako zbiór zrzutów ekranu:

- Strona główna przedstawiona jest na rysunku 4.25.
- Podstrona wykresów przedstawiona jest na rysunku 4.27.
- Podstrona zadań automatycznych przedstawiona jest na rysunku 4.26.
- Podstrona dziennika zmian stanu gniazd przedstawiona jest na rysunku 4.28.
- Podstrona konfiguracji przedstawiona jest na rysunku 4.29 i 4.30.

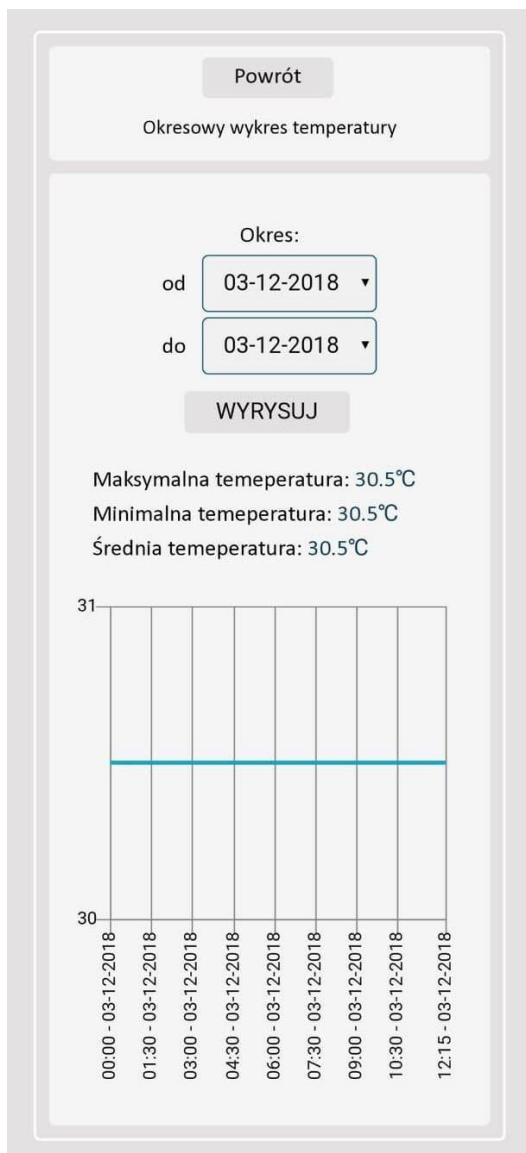
⁸ Metatag – dostarcza informacji na temat dokumentu HTML. Dane te są przetwarzane przez przeglądarki, boty indeksujące oraz inne usługi [18].



Rys. 4.25 Strona główna



Rys. 4.26 Podstrona zadań automatycznych



Rys. 4.28 Podstrona wizualizacji

Powrót

Dziennik zmian stanów gniazd

ID logu	adapter	gniazdo	kto
1071	SYMULOWANA LISTWA, RPI	Gniazdo testowe 4	wojtek
stan gniazda		czas	data
włączenie		05:34	02-12-2018

ID logu	adapter	gniazdo	kto
925	SYMULOWANA LISTWA, RPI	Gniazdo testowe 1	admin
stan gniazda		czas	data
wyłączenie		10:49	11-11-2018

ID logu	adapter	gniazdo	kto
904	SYMULOWANA LISTWA, RPI	Gniazdo testowe 3	admin_wojtek
stan gniazda		czas	data
włączenie		08:42	06-11-2018

ID logu	adapter	gniazdo	kto
903	SYMULOWANA LISTWA, RPI	Gniazdo testowe 3	admin_wojtek

Rys. 4.29 Podstrona rejestru gniazd

Powrót

Aktualna data systemu: 23-03-2019
Aktualny czas systemu: 21:47:48

Stan systemu

Listwa obsługiwana częściowo

Status gniazd: **zablokowane ręcznie**

ID listwy: 1
Pokój: SYMULOWANA LISTWA
Miejsce: RPi

Stan połączenia: **slabe**
Moc nadawcza listwy: HIGH
Liczba prób połączenia: 99

Dostępne opcje dla listwy:

- Wyłącz obsługę listwy
- Anuluj aktywne zadania
- Zatrzymaj aktywne zadania
- Dezaktywuj listwę
- Zmień nazwę

Zarządzenie połączeniem:

- Przypisz adres (do dodania)
- Ustaw liczbę prób połączenia
- Zmień moc listwy

Czujnik temperatury włączony

Sensor ID: 1
DS18B20
Dane: 31°C
Czas i data: 12:18:27, 03-12-2018

- Wyłącz sensor
- Usuń sensor
- Zmień nazwę

Czujnik światła wyłączony

Sensor ID: 6
fotorezystor
Dane: 21%
Czas i data: 12:18:27, 03-12-2018

- Włącz sensor
- Usuń sensor
- Zmień nazwę

Socket ID: 1
Gniazdo testowe 1
Gniazdo włączone
Kontrola przez menadżer zadań

- Usuń gniazdo
- Zmień nazwę

Socket ID: 2
Gniazdo testowe 2
Gniazdo włączone
Gniazdo sterowane ze strony

- Usuń gniazdo
- Zmień nazwę

Socket ID: 3
Gniazdo testowe 3
Gniazdo włączone
Gniazdo sterowane ze strony

- Usuń gniazdo
- Zmień nazwę

Socket ID: 20
Gniazdo testowe 4
Gniazdo włączone
Gniazdo sterowane ze strony

- Usuń gniazdo
- Zmień nazwę

Rys. 4.29 Podstrona konfiguracji

Restart systemu

Ustawienia stacji matki

Zmiana godziny i daty

Podaj nową godzinę:
HH:mm:ss
Zmień godzinę

Podaj nową datę:
RRRR-MM-DD
Zmień datę

Zmiana mocy nadajnika

Wybierz moc nadawczą:
Niska
Zmień moc

Ustawienia użytkowników

Dodawanie użytkownika

Podaj nazwę użytkownika:
Podaj hasło użytkownika:
Powtórz hasło:
Typ konta:
uzytkownik
Dodaj użytkownika
Zmień hasło

Zmiana uprawnień

Wybierz użytkownika:
admin [admin, 28-11-2018, 0
Typ konta:
uzytkownik
Zmień uprawnienia i odblokuj
Zablokuj użytkownika

Ustawienia konta

Zmiana hasła

Podaj nowe hasło:
Powtórz nowe hasło:
Zmień hasło

Zmiana czasu odświeżania strony

Domyślne odświeżanie wynosi: 60 sekund
Aktualny czas: 60 sekund
Podaj czas odświeżania strony głównej i konfiguracji:
Zmień czas

Rys. 4.30 Podstrona konfiguracji c.d.

51

5. Sterowanie listwami

W rozdziale tym opiszemy sposoby, w jakie możemy zaprogramować listwę. Algorytmy postępowania będą dotyczyły działania w pełni automatycznego lub poprzez użytkownika.

Scenariusz 1

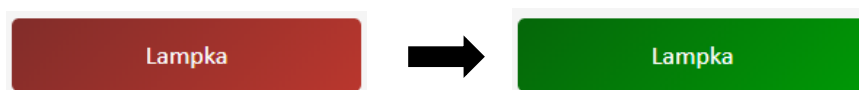
Chcemy załączyć/wyłączyć pojedyncze gniazdo na listwie poprzez komputer/telefon.

Krok 1: Łączymy się z siecią lokalną Wi-Fi, do której jest podłączony system.

Krok 2: Przechodzimy na stronę systemu (znane jest IP strony, np. 192.168.0.100, jeśli jest to telefon dopisujemy do adresu /mobile lub klikamy „WERSJA MOBILNA STRONY”).

Krok 3: Logujemy się do systemu,

Krok 4: Wybieramy dane gniazdo w listwie i jeśli jest czerwone/zielone to klikamy na nie. Jeśli mamy połączenie to gniazdo się załączy/wyłączy na listwie (rys. 5.1).



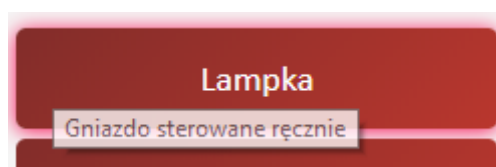
Rys. 5.1 Zmiana stanu gniazda

Scenariusz 2

Chcemy przełączyć gniazdo w momencie, gdy jest ono sterowane ręcznie.

Kroki 1-3 są identyczne jak w scenariuszu 1.

Krok 4: Klikamy na gniazdo. Nic się nie dzieje, gdyż listwa jest manualnie przełączona w tryb włączenia wszystkich gniazd (rys. 5.2). Informacja o stanie gniazd zmienia wartość z *sterowalne* na *zablokowane*. Żadne zadanie w tym stanie się nie wykona.



Rys. 5.2 Status zadania

Scenariusz 3

Chcemy zautomatyzować gniazdo, aby działało według jednego z warunków.

Kroki 1-3 są identyczne jak w scenariuszu 1.

Krok 4: Przechodzimy do opcji listwy „Zadania automatyczne”.

Krok 5: Wybieramy typ zdania. Mamy do wyboru jeden z poniższych:

- Wyłącz o wskazanej godzinie
- Włącz o wskazanej godzinie
- Wyłącz, gdy temperatura wzrośnie powyżej
- Włącz, gdy temperatura wzrośnie powyżej
- Wyłącz, gdy temperatura spadnie poniżej
- Włącz, gdy temperatura spadnie poniżej

- Wyłącz, gdy natężenie światła wzrośnie powyżej
- Włącz, gdy natężenie światła wzrośnie powyżej
- Wyłącz, gdy natężenie światła spadnie poniżej
- Włącz, gdy natężenie światła spadnie poniżej

Krok 6: Podajemy wartość warunku. Dla odpowiednich zadań wymagany jest odpowiedni format warunku:

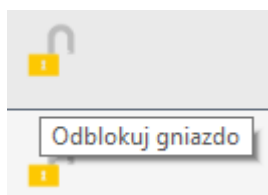
- *HH:mm*, dla zadania czasowego.
- *XX*, dwucyfrowa wartość natężenia światła w %, 99% odpowiada światłu słonecznemu w bezchmurny dzień, wartość ta musi być z zakresu od 0 do 99.
- *TT,tt*, wartość temperatury można podać w formacie zmiennoprzecinkowym lub nie, wartość ta musi być z zakresu od 0 do 99, dokładność do dwóch miejsc po przecinku.

Krok 7: Wybieramy interesujące nas gniazdo spośród 4 dostępnych w listwie.

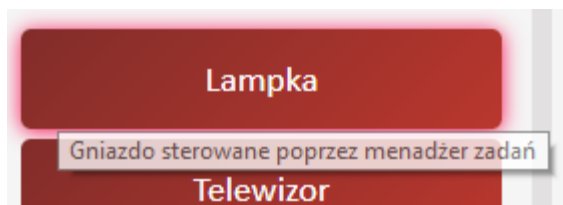
Krok 8: Wybieramy typ powtarzalności gniazda.

Krok 9: Klikamy dodaj zadanie.

W tym momencie zadanie zostanie dodane do listy „Menadżer zadań listwy” poniżej panelu dodawania. Gniazdo jest automatycznie aktywne oraz zablokowane dla ręcznej zmiany stanu. Informacja, o tym czy jest ono zablokowane jest przedstawiana w dwóch miejscach: w tabeli w rubryce blokada (rys. 5.3) oraz na liście gniazd na stronie głównej (rys. 5.4).



Rys. 5.3 Blokada gniazda na liście zadań



Rys. 5.4 Blokada gniazda na stronie głównej

Jeśli podano poprawnie warunek oraz wykorzystywany czujnik jest włączony to w rubryce błąd nic się nie pojawi, a zadanie będzie funkcjonować.

Jeżeli chcemy sprawdzić czas ostatniej aktualizacji lub kiedy zadanie zostało wykonane, możemy to zrobić przechodząc na stronę główną i wchodząc na podstronę „Podgląd zadań automatycznych”, który wyświetla listę zadań w całym systemie. Mamy tutaj podgląd uruchomionych oraz anulowanych bądź zakończonych już zadań wraz z ich statusem (rys. 5.4).

ID	adapter	gniazdo	typ zadania	warunek	cykliczne	uzytkownik	czas	data	informacja	komentarz
183	614, za tóżkiem	Telewizor	Wyłącz o wskazanej godzinie	12:12	0	wojtek	07:28	06-01- 2019	Zakończono: 12:12, 06- 01-2019	brak

Rys. 5.5 Status zadania

Scenariusz 4

Chcemy zautomatyzować gniazdo, aby działało według jednego z warunków w danym przedziale czasu.

Kroki 1-5 są identyczne jak w scenariuszu 3.

Krok 6: Wybieramy opcje kontroli czasowej i podajemy przedział czasu pracy zlecenia. Należy zauważyć, że opcja ta traci sens dla przypadku, gdybyśmy wybrali w kroku 5 opcję włączenia/wyłączenia czasowego.

Krok 7: Podajemy wartość warunku. Dla odpowiednich zadań wymagany jest odpowiedni format warunku:






- *HH:mm*, dla zadania czasowego.
- *XX*, dwucyfrowa wartość natężenia światła w %, 99% odpowiada światłu słonecznemu w bezchmurny dzień, wartość ta musi być z zakresu od 0 do 99.
- *TT,tt*, wartość temperatury można podać w formacie zmiennoprzecinkowym lub nie, wartość ta musi być z zakresu od 0 do 99, dokładność do dwóch miejsc po przecinku.

Krok 8: Wybieramy interesujące nas gniazdo spośród 4 dostępnych w listwie.

Krok 9: Wybieramy typ powtarzalności gniazda.

Krok 10: Klikamy dodaj zadanie.

Różnica jaka się pojawi względem poprzedniego scenariusza to pojawienie się symbolu zegara, na którym po najechaniu pokaże się wybrany przedział czasu (rys. 5.6).

242		Lampka nocna	Włącz o wskazanej godzinie	00:00	powtarzalne	Wojtek	23:40 04-06-2019		Usuń
240		Lampka nocna	Wyłącz, gdy natężenie światła wzrośnie powyżej	10	powtarzalne	wojtek	21:04 28-05-2019		 <div> Początek: 20:00 Koniec: 00:00 </div>

Rys. 5.6 Status zadania

6. Uruchamianie systemu

6.1. Środowisko Raspian

Pierwszym etapem pracy przy uruchomieniu systemu jest wgranie systemu *Raspian* do Raspberry Pi. W tym celu należy przekopiować odpowiednie pliki na kartę pamięci, z której będzie instalowany system.

Po instalacji systemu należy ustawić strefę czasową oraz wgrać odpowiednie biblioteki oraz oprogramowanie do obsługi wszystkich funkcji systemu, a są to:

- Usługa serwera HTTP Apache, robimy to poprzez polecenia [19]:

```
sudo apt-get install apache2 -y
```

- Biblioteka PHP :

```
sudo apt-get install php libapache2-mod-php -y
```

- Biblioteka MySQL++ [20]:

Należy zainstalować pakiet *default-libmysqlclient-dev* dla wersji armhf, poprzez pobranie go ze strony: <https://packages.debian.org/stretch/default-libmysqlclient-dev> oraz wypakowanie i zainstalowanie poleceniem[21]:

```
sudo apt install ./[nazwa].deb
```

- Baza danych MariaDB i PhpMyAdmin (hasło *rpiipacs*):

```
sudo apt-get install mysql-server --fix-missing
sudo apt-get install phpMyAdmin
```

oraz dodać do konfiguracji apache tę linijkę:

```
# phpMyAdmin Configuration
Include /etc/phpmyadmin/apache.conf
```

Następnie należy dodać użytkownika do bazy danych i nadać mu wszystkie przywileje poprzez komendy w MySQL:

```
grant all privileges on *.* to pi@localhost identified
by 'rpiipacs' with grant option;
```

```
i
```

```
flush privileges;
```

Użytkownik *pi*, hasło *rpiipacs*, takie samo jak do samego Raspberry Pi.

- Biblioteka RF24 [22]:

```
git clone https://github.com/nRF24/RF24.git
cd RF24
sudo make install
```

Do przesyłania plików pomiędzy stacją roboczą, a Raspberry Pi posłuży program *WinSCP*, który wykorzystuje protokół SCP używający szyfrowania SSH. Domyślnie SCP na RPi jest włączone, co pozwala bez dostępu do wyświetlacza i klawiatury na szybkie przesłanie plików.

Kolejnym krokiem jest napisanie programu, który będzie nam uruchamiał system przy starcie, abyśmy mieli ciągłość działania usługi, w przypadku braku prądu lub skoku napięcia. Program zostanie opisany przy tłumaczeniu problemów (roz. 6.3.2) i nazywa się *ipacsStart*, wykorzystano także program *Crontab* do zarządzania uruchamianiem podczas bootowania.

Od tej pory możliwe jest pisanie programów w języku C++ i ich kompilowanie, dzięki wbudowanemu w system Linux kompilatorowi G++.

Kompilacja możliwa jest poprzez komendę:

```
g++ -Wall -Ofast -Wextra -mfpu=vfp -mfloat-abi=hard -march=armv6zk
-mtune=arm1176jzf-s -lmysqlclient -lrf24 -g ipacs.cpp Task.cpp -o ipacs
```

Interesującymi nas parametrami są: *-lmysqlclient* oraz *-lrf24*, które dodają ścieżki biblioteki do kompilacji, *-g* czyli tryb, który pozwala debugować program poprzez aplikację *gdb*, *xxx.cpp* to plik źródłowy, który jest kompilowany oraz plik wyjściowy czyli *-o*. Pozostałe parametry nie zostaną opisane i służą do poprawnej kompilacji na RPi.

Skompilowane programy uruchamiamy poprzez komendę:

```
sudo ./ipacs &
```

Jeśli chcemy zmienić argumenty uruchamiania, to zamieniamy znak & na dowolny z argumentów opisanych wcześniej (roz. 4.3.2).

Aby zatrzymać program w trakcie jego działania w tle, można to zrobić na dwa sposoby, manualnie, poprzez odszukanie id procesu (komenda *ps* i *grep*) lub poprzez tekstowy menadżer zadań *htop*. Wymaga on wpisania w filtr nazwy procesu i wciśnięciu przycisku od zabicia procesu (później typu zamknięcia programu). Jest to szybsze rozwiązanie, które dodatkowo pozwala monitorować zużywane zasoby.

Początkowo wystarczy podłączyć kabel Ethernet do RPi, aby klient DHCP wynegocjował warunki połączenia i otrzymał adres IP. Do przypisania stałego adresu IP, należy poznać adres MAC karty sieciowej, aby to sprawdzić należy wykonać polecenie *ifconfig*.

6.2. Środowisko stacji roboczej

Na komputerze należy zainstalować program *WinSCP*, *ArduinoIDE* oraz *Visual Code*, jako edytor C++ (poprzez wtyczkę). Do sterowania RPi zdalnie możemy użyć aplikacji z interfejsem graficznym, czyli VNC Viewer działającego poprzez protokół VNC (domyślnie włączony) lub *PuTTY* z interfejsem tekstowym. Ten drugi pozwala na komunikację różnymi protokołami komunikacyjnymi. Z racji tego, że interfejs graficzny na RPi nie jest w projekcie potrzebny, wybrana została druga opcja, a system w wersji ostatecznej będzie miał usunięty interfejs graficzny, aby zaoszczędzić zasoby i zmniejszyć konsumpcję prądu.

W programie *Visual Code* możemy poprzez wtyczki edytować pliki typu cpp jak i css, html, php itd. Jest on przyjemną formą edytora, która może być rozszerzona o niezbędne dodatki w każdym momencie, a ich baza ciągle się powiększa.

Należy także wejść na stronę administracyjną routera, aby przypisać RPi stały adres IP, po którym system będzie odpowiadać. Taką konfigurację przedstawia rysunek 6.1.

Rezerwacja adresów				
ID	Adres MAC	Zarezerwowane IP	Status	Zmień
1	B8-27-EB-A1-70-C0	192.168.0.100	Włączony	Edytuj Usuń
<input type="button" value="Dodaj nowy..."/> <input type="button" value="Włącz wszystkie"/> <input type="button" value="Wyłącz wszystkie"/> <input type="button" value="Usuń wszystkie"/>				

Rys. 6.1 Konfiguracja routera

6.3. Problemy związane z systemem

W rozdziale tym, zajmiemy się omówieniem problemów, które spowodowały dłuższy przestój w postępach pracy. Projekt jest dość rozbudowany i nierozsądnie byłoby zakładać, że wszystko będzie zaplanowane z góry i obędzie się bez trudności. Zbyt małe doświadczenie autora w dziedzinach elektroniki i programowania musi doprowadzić do pewnych przeszkód. Można je pokonać przez znalezienie rozwiązania lub sposobu na obejście problemu. W systemie zostały wykorzystane obie drogi, co raczej wynikało z natury problemu i braku odpowiedniej wiedzy autora niż z braku zaangażowania.

6.3.1. Problem transmisji pakietów

Problem ten składa się z kilku wymienionych niżej trudności:

- I. *Problem transmisji w obrębie jednego pokoju, gdzie istnieje wiele ech i odbiornik nie jest przystosowany do ich filtracji.*

Jak się okazało odbiornik potrafił odebrać kopię wiadomości, ale działo się tak tylko przy małych odległościach od transponderów (mniej niż 3 metry). Rozwiązaniem problemu było dodanie flagi odebrania wiadomości. Wszystkie wiadomości są odbierane, ale jeśli już zostanie odebrana od danego adaptera (flaga została ustawiona) to echa są ignorowane. Bufor wiadomości po każdym odebraniu jest czyszczony.

- II. *Problem poprawnej wiadomości.*

Jak się okazało, przy wykorzystaniu CRC16 wiadomości potrafiły przejść z nadajnika do odbiornika niepoprawnie i w programie przekłamate były znaki. Poprawność transmisji przy użyciu CRC jest na poziomie 100%, problem raczej leży po stronie SPI i tam według autora zachodzi przekłamanie. Rozwiązaniem problemu jest sprawdzanie składni wiadomości, gdzie pomiędzy dane paczki danych wstawiany jest znak „podłogi”, czyli „_”. Musi on wystąpić w odpowiednich miejscach ramki. Autor pracował w miejscu dużych zakłóceń, od urządzeń elektrycznych po mikrofalówki i wiele urządzeń Wi-Fi, nie wspominając o bliskości stacji bazowych operatorów telekomunikacyjnych.

III. *Problem weryfikacji stanu połączenia.*

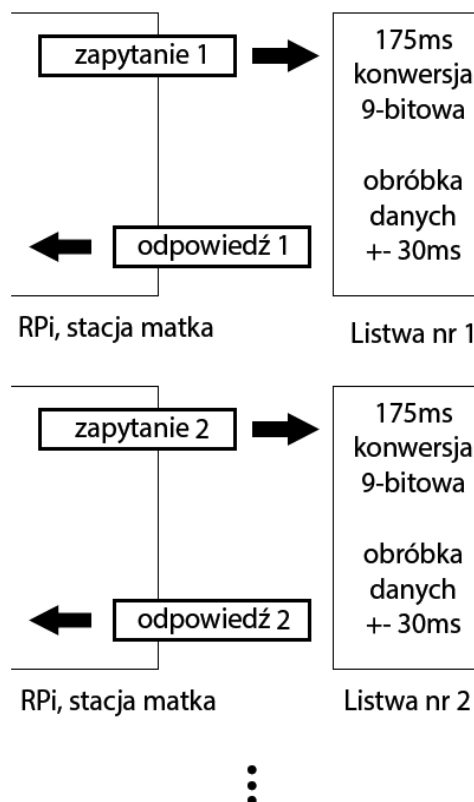
W trakcie pracy autor doszedł do konkluzji, że jeśli nie ma połączenia z daną listwą, to musi to być w jakiś sposób przedstawione na stronie. Dodatkowo, jeśli nie wszystkie pakiety dochodzą to taka informacja o „słabym połączeniu” także byłaby przydatna do poszukiwania optymalnej mocy listwy. Rozwiązaniem było dodanie do programu licznika nieudanych prób połączenia, które znaczą tyle co odpytanie i odpowiedź, jeśli odpowiedź zostanie otrzymana w zadanym czasie to licznik jest zerowany. Próg „słabego połączenia” i „braku połączenia” każdy użytkownik może sam zdefiniować poprzez panel konfiguracyjny.

IV. *Problem czasu odpowiedzi listwy*

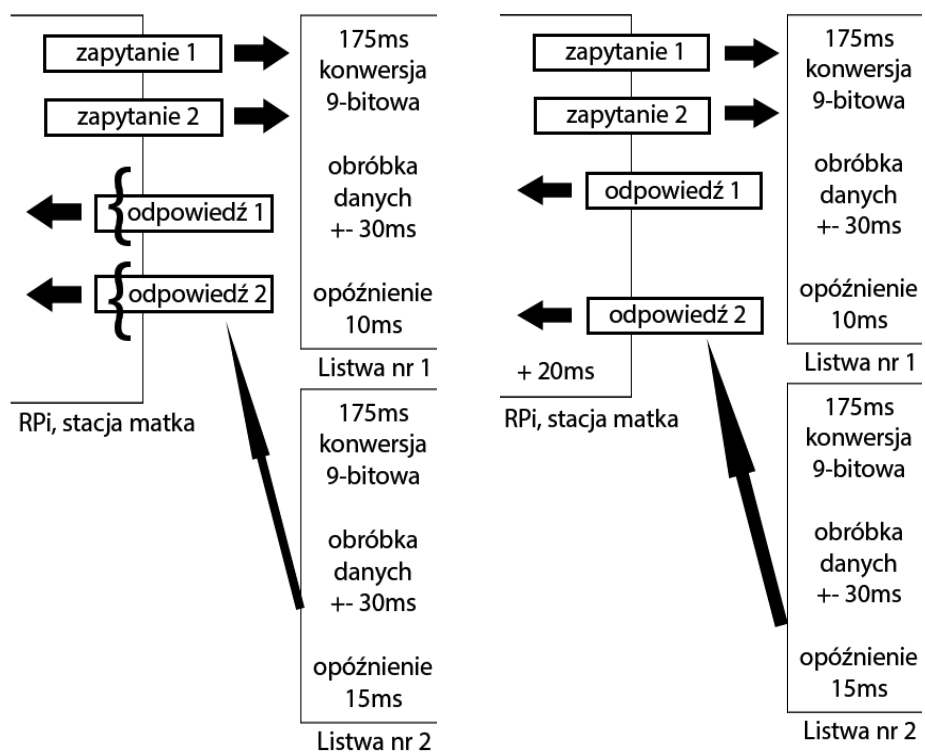
Problem ten stał się jednym z większych, gdyż doczekał się aż 3 rozwiązań, przy czym każde następne było lepsze od poprzedniego. Chodzi tutaj o to, że czujnik temperatury potrzebuje czasu na przetworzenie danych termicznych (około 175ms, dla 10bit). Aby system działał w pewien sposób synchronicznie (co daje kontrolę) nie możemy nakazać Arduino, aby odpytywało czujnik cały czas i w momencie zakończenia konwersji sprawdzało, czy przyszło zapytanie. Przy małej liczbie listw w systemie Arduino nadawałoby asynchronicznie, nie znalazłbyśmy momentu odpowiedzi, a Raspberry przez system operacyjny generuje opóźnienia w pracy naszego systemu i możemy trafić w moment, gdzie Raspian jest zajęty obsługą procesu niezwiązanego z projektem. Dodatkowo, przy dużej liczbie listw, bufor, który ma 3 rejestry zapełniłby się i istniałaby możliwość, że adapter nie odbierze zaadresowanej dla siebie wiadomości, jeśli w tym momencie wykonywałby przetwarzanie.

Rozwiązania były następujące:

- 1) Wysyłanie i odbiór wiadomości jedna po drugiej. Dla jednej listwy było to dobrym rozwiązaniem, problem powstał wówczas, gdy autor dodał drugą listwę symulacyjną. Pojawiło się opóźnienie, przy reakcji na kliknięcie na stronie w gniazdo. Dodanie każdej kolejnej listwy wydłużało to opóźnienie o kolejne (w przybliżeniu, przy włączonej transmisji szeregowej) 300ms. Prezentuje to rysunek 6.2.
- 2) Wysyłanie wiadomości w szeregu wraz z określonym opóźnieniem, po czym przydzielenie okien czasowych na odpowiedzi z listw. Jeśli wiadomość nie przyszła w danym oknie to jest ignorowana (może być echem). Rozwiązanie to, z zamysłu dobre, stało się problematyczne z racji losowości rozrzutu czasowego pracy systemu. Trudno było dobrać optymalne długości okien czasowych (rys. 6.3). Przy dość dużych oknach i opóźnieniach system pracował tak jak powinien, lecz nie było to optymalne rozwiązanie. Pozostało ono jednak takim, do momentu znalezienia kolejnego, lepszego.
- 3) Ostatecznym rozwiązaniem, jak się okazało prostym, zostało wysyłanie wiadomości w szeregu, ze stałymi opóźnieniami względem siebie oraz z czasem oczekiwania na wszystkie wiadomości jednocześnie i nadawaniem flag dla ech. Czas oczekiwania na wiadomości został wyliczany względem ostatniej wiadomości w kolejce. Przy odpowiednim czasie zwłoki wszystkie wiadomości zdążą dojść do stacji matki. Przedstawienie problemu na rysunku 6.4.



Rys. 6.2 Rozwiązanie nr 1 problemu opóźnienia pakietów



Rys. 6.3 Rozwiązanie nr 2 problemu opóźnienia pakietów

Rys. 6.4 Rozwiązanie nr 3 problemu opóźnienia pakietów

6.3.2. Problem uruchamiania systemu operacyjnego

Problem ten wymagał od autora nie lada wysiłku oraz czasu, aby go rozwiązać. Rozwiązanie jednak nie daje oczekiwanej satysfakcji. Program systemu powinien się uruchamiać wraz ze startem systemu operacyjnego *Raspian*, dlatego też została dodana odpowiednia linijka do Crona⁹ poprzez komendę *crontab -e*:

```
@reboot /home/pi/Desktop/ipacs_programy/start_ipacs
```

Program *ipacs_start* uruchamia wszystkie programy systemu:

```
sleep_until(system_clock::now() + 10000ms);
system("sudo /home/pi/ipacs_program/insertSensorData_timeIntervalExecute
&");
system("sudo /home/pi/ipacs_program/ipacs &");
system("sudo /home/pi/ipacs_program/taskManager &");
```

Kłopot jest w tym, że jak się okazało Cron uruchamia się dużo szybciej na liście uruchamiania niż baza danych. Co za tym idzie? Jeśli nasz program spróbuje uruchomić się szybciej, to nie nawiąże połączenia z bazą i się wyłączy. Rozwiązaniem banalnym, lecz niesatysfakcjonującym jest dodanie prostego opóźnienia widocznego na kodzie powyżej. Dla bezpieczeństwa wynosi ono 10 sekund i jest wystarczające do RPi w wersji trzeciej.

6.3.3. Problem dostępowy (Access Point)

W pewnym momencie przyszedł pomysł, żeby wykorzystać bezprzewodową kartę sieciową jako punkt dostępowy. Użytkownicy sieci łączyliby się do niej jak do zwykłej sieci WiFi i przechodzili do strony internetowej poprzez odpowiednie IP.

Niestety jak się okazało, praca programu *RaspAP*, który pozwala na utworzenie takiej konfiguracji zawiodła. Interfejs dostępowy raz się uruchamiał w trybie dostępu, a raz jako klient WiFi. Autor, z racji małego doświadczenia, nie potrafił sobie z tym faktem poradzić i zaniechał tego rozwiązania.

Co ciekawe, mimo pełnego odinstalowania i przywrócenia starych plików konfiguracyjnych interfejs dalej się co jakiś czas uruchamia samoczynnie. Pokazuje to, jak słabe jest to rozwiązanie.

6.3.4. Problem sterowania mocą

Listwa nie może mieć na sztywno ustawionej mocy transmisji, gdyż powodowałoby to, że każda zmiana położenia w przypadku utraty zasięgu zmuszałaby administratora do przeprogramowywania urządzenia.

Rozwiązaniem stało się przesyłanie wiadomości zarządzania o id 4, której budowa wygląda następująco:

⁹ Cron – jedno z bardzo użytecznych narzędzi wszystkich Unixo-podobnych systemów. Pozwala planować wykonanie komend w czasie [23].

4_XX_P, gdzie XX to adres adaptera, a P to wartość mocy licząc od 0 do 3 włącznie.

Wysyłana jest ona do 20 razy, w przypadku braku nadejścia wiadomości potwierdzającej odbiór (ACK).

Problem pojawia się, gdy listwa odbiera wiadomości, ale potwierdzenia nie dochodzą do RPi. Oznacza to, że moc zostaje zmieniona, ale jeśli potwierdzenie nie dojdzie, to w systemie widnieje ona na poprzednim poziomie.

6.3.5. Problem zadań kontrolowanych czasowo

Jak się z czasem okazało, do elastycznej pracy systemu przydałaby się dodatkowa funkcjonalność, to znaczy zlecenia, które mogłyby działać tylko w danym przedziale czasu, np. tylko w nocy.

Rozwiązań teoretycznych tego problemu było wiele, ale żadne nie było proste. Program nie był pisany pod taką ewentualność. Aby zminimalizować zmiany w kodzie, należało dodać parę rzeczy w bazie danych oraz na stronie. Są nimi: dodatkowe kolumny w encji *socket_tasks*: *task_time_controlled*, *task_time_on*, *task_time_off* oraz miejsce na wpisanie tych danych na stronie i ich sformatowanie.

Ostateczne rozwiązanie zostało już opisane w rozdziałach wcześniej. Kod do tego potrzebny jest dość obszerny, lecz nie wymaga zmiany czegokolwiek już istniejącego, a jedynie dodania paruset, redundantnych linii jako kolejny warunek dla menadżera.

6.3.6. Problem utraty połączenia

Początkowo zamierzono, aby w przypadku utraty połączenia listwa wyłączała obsługę. W trakcie testowania okazało się, że moduły mają na tyle małą moc, że sygnał potrafił zanikać na parę minut, co ogranicza pracę systemu. Dlatego, aby automatyzacja działała, problem rozwiązano w prosty sposób, obsługa listwy jest zawieszana do momentu automatycznego odzyskania połączenia. Nie ma możliwości ręcznego wyłączenia obsługi listwy, tylko odpytywania, co wiąże się z wyłączeniem obsługi. Rozwiązanie to sprawdza się np. w momencie utraty próbki spowodowanej brakiem komunikacji, będzie pobrana próbka z okolicy tego czasu (ostatnia odebrana próbka).

6.4. Etapy prac nad projektem

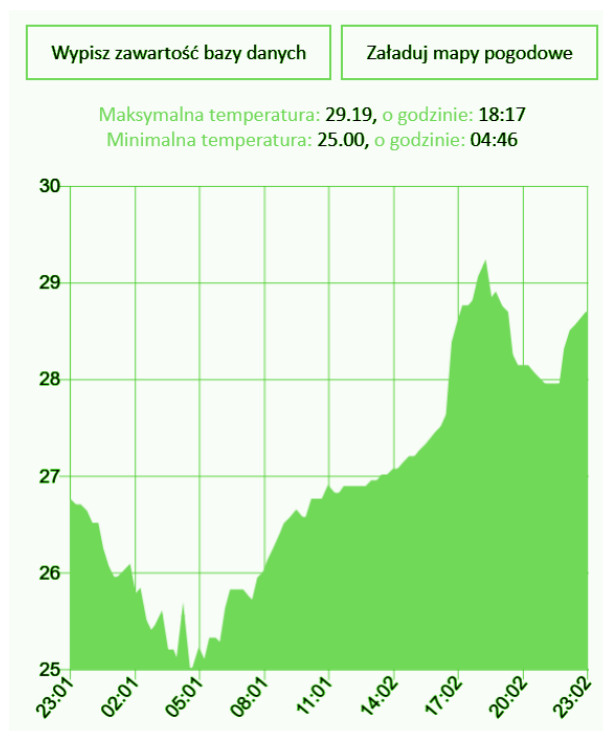
W rozdziale tym zostaną w skrócie opisane etapy prac nad projektem oraz jakie wyzwania jakie zostały postawione na danym etapie. Praca została rozłożona w czasie, ale każde zadanie wymagało skupiania, aby wykonać je jak najefektywniej. Przerwy trwały nawet miesiąc, gdyż wymagane były bieżące testy oraz świeże podejście do sprawy.

Etap 1

Autor tuż przed wybraniem tematu pracy zainteresowany był możliwościami Raspberry Pi, dlatego też podłączył do urządzenia czujnik temperatury i zaczął zgłębiać wiedzę dotyczącą portów GPIO oraz protokołów komunikacyjnych. Początkowo dane przy użyciu protokołu OneWire były zapisywane do pliku.

Napisana została strona do wyświetlania danych z pliku, Program napisany jeszcze w Pythonie został zmodyfikowany tak, aby rejestrować kilka pomiarów w odstępach czasu,

co umożliwia wyrysowanie wykresu (rys. 6.5). Okazało się, że odczyt z pliku trwa dość długo i rozwiązanie to nie jest optymalne. Autor doszedł do wniosku, że lepszą opcją jest użycie bazy danych, gdyż zapis i odczyt można zrównoleglić i jest on dużo szybszy. Do tego wymagane było przepisanie programu do języku C++.



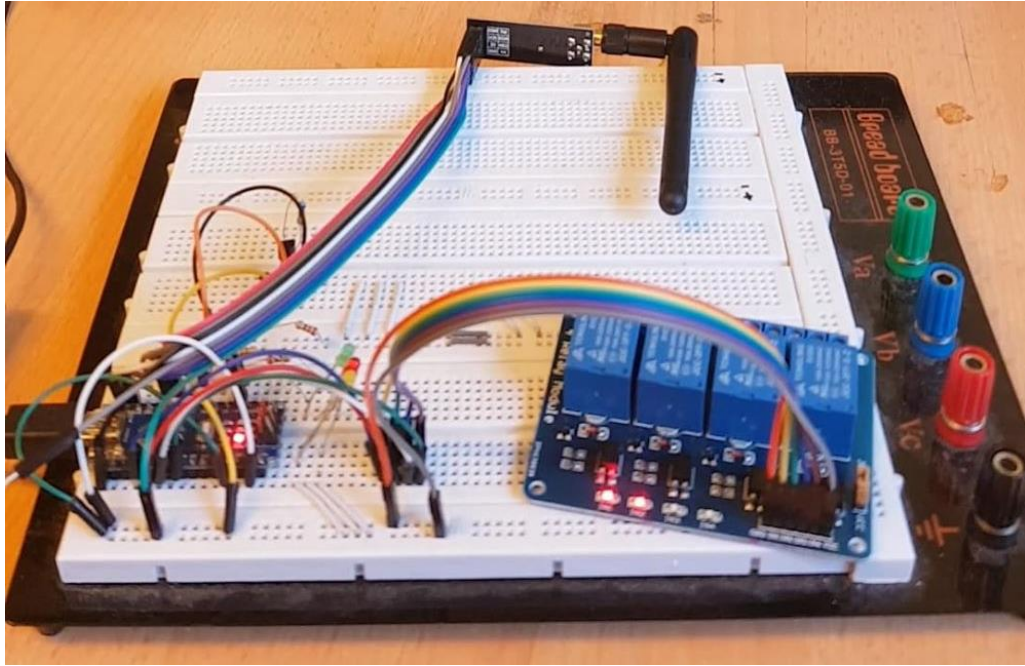
Rys. 6.5 Pierwsze wykresy temperatury

Etap 2

W tym momencie pojawił się pomysł jak rozwinąć mały projekt do skali wcześniej już opisanej. Autor usłyszał o modułach NRF24L01 i ich możliwościach i postanowił spróbować na nich coś zbudować. Podłączono Arduino z modulem na płytce stykowej oraz z RPi poprzez najzwyklejsze kable „jumpery”. Po wgraniu niezbędnego oprogramowania doszło do prób komunikacji. Pierwsze próby polegały na komunikacji jednostronnej z RPi do Arduino i na odwrót. Testy zakończyły się niepowodzeniem. Zajęło to chwilę, zanim okazało się, że parametry nadajników nie były jednakowe. Po udanej komunikacji można było przerobić program, aby wysyłał dane o temperaturze z Arduino. RPi odbierało dane i wysyłało je na serwer MariaDB.

Etap 3

Komunikacja działała na modułach w wersji bez dołączonych anten (mikropaskowych), jednak zasięg okazał się dość słaby. Podjęto decyzję o zamówieniu modułów z antenami unipolowymi. Program został wzbogacony o możliwość wysyłania stanów gniazd w jedną stronę i odbioru danych o temperaturze i natężeniu światła. Podłączony został moduł przekaźników, aby można było symulować sterowanie gniazdami (rys. 6.6). Pierwsze wersje strony nie były zbyt ładne ani intuicyjne, dlatego że miały po prostu działać. Autor zaniechał chęci pokazywania tych wczesnych prototypów interfejsu.



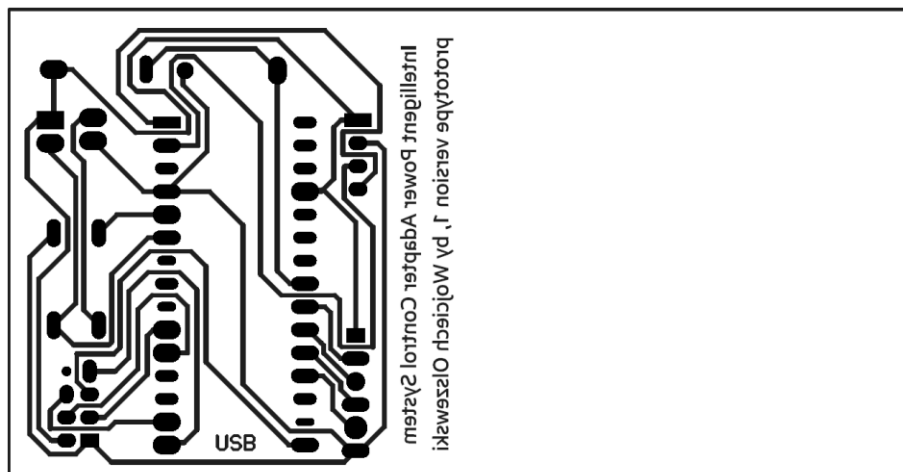
Rys. 6.6 Układ listwy na płycie stykowej

Etap 4

Skoro system już w pewien sposób funkcjonował, można było zabrać się za część programistyczną i dodać funkcje, które były w założeniach projektowych. Etap ten polegał na oprogramowaniu strony oraz stworzeniu programu sieci, dlatego też zajął немало czasu. Efektem tego były zadania automatyczne oraz funkcjonalny panel konfiguracyjny. Rozwiązane w tym czasie były też problemy wspomniane w rozdziale 6.3 i dodana została listwa symulacyjna, która imitowała do kilku listw jednocześnie.

Etap 5

Na tym poziomie brakowało zwartego układu, aby można było łatwo przenosić system. Należało zaprojektować oraz wykonać prototyp płytki drukowanej. Za pomocą technik wytrawiania druku udało się taki prototyp wykonać w dość dobrej jakości. Projekt przedstawia rysunek 6.7.



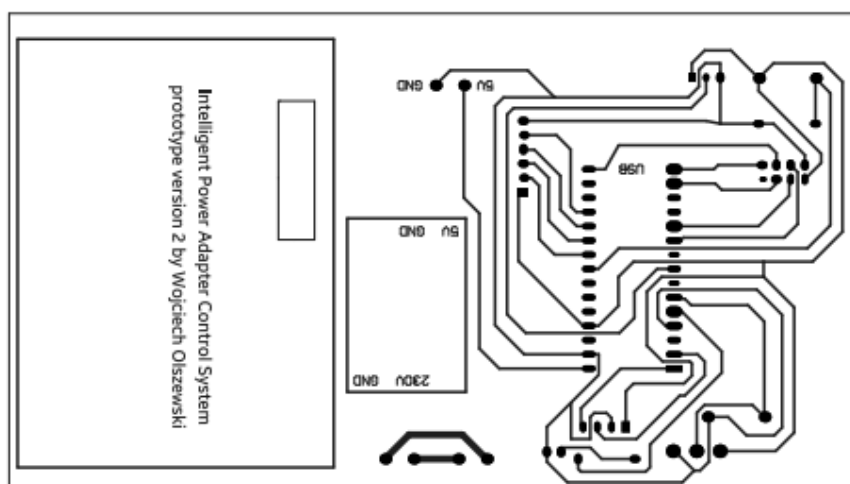
Rys. 6.7 Pierwszy prototyp PCB układu listwy

Etap 6

Nastąpił etap dopracowania projektu pod kątem programistycznym. Do tej pory było to podejście robocze na zasadzie „ma działać”, od tego momentu rozpoczęło się tworzenie wersji końcowych oprogramowania. Szata wizualna została zmieniona tak, aby interfejs był przyjazny oku. Po rozmowie z kilkoma osobami, zostały uproszczone niektóre elementy interfejsu oraz dodane zostały komentarze jak i instrukcja. W trakcie testowania i „zabawy” okazało się, że przydałyby się dodatkowe funkcje, np. wyczyszczenie błędów zadań automatycznych listwy. Kod programu także otrzymał pewien lifting pod kątem wyrzucenia niepotrzebnego kodu.

Etap 7

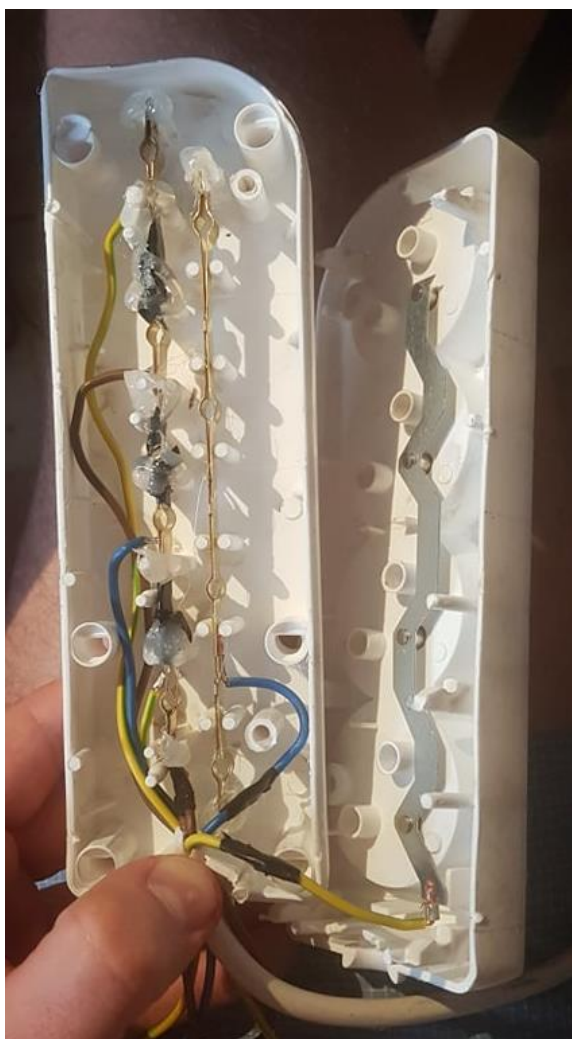
Do tej pory system działał czysto symulacyjnie, gdyż nie zasilano urządzeń za jego pomocą. Testy wykazały, że działa on na tyle stabilnie, że można przejść na poważniejszy poziom, czyli wykorzystanie prądu 230V. Do tego celu zamówiona została zawczasu przetwornica 230V na 5V, która także została przez autora sprawdzona. Po zaprojektowaniu odpowiedniej płytki PCB (rys. 6.8) wykonano ją, lecz ta wersja z braku czasu i szczęścia nie wyszła dość dobrze, mimo to dzięki wysiłkowi autora i ręcznym poprawkom działała w 100%.



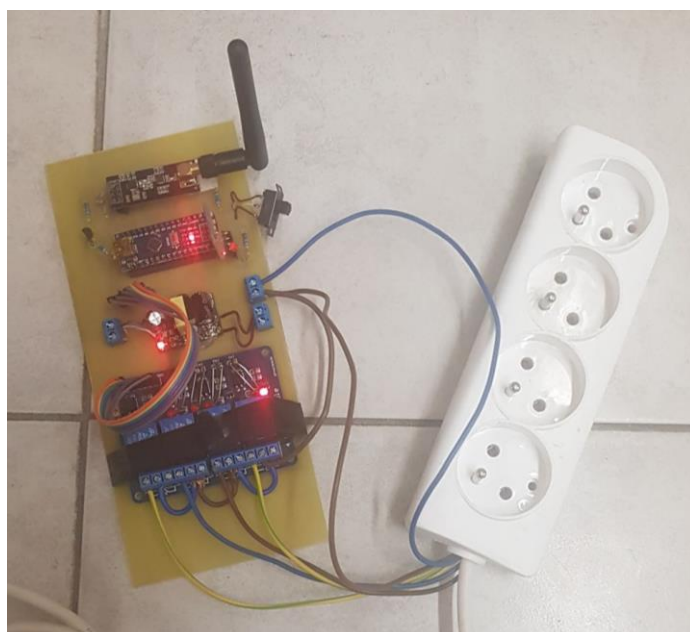
Rys. 6.8 Drugi prototyp PCB układu listwy

Etap 8

Po wykonaniu drugiego prototypu posiadającego wyprowadzenia ARK do podłączenia prądu 230V możliwe się stało przerobienie układu przedłużacza, aby gniazda mogły być sterowane. Wykonanie to polegało na pocięciu blaszki, która była zamontowana w środku i łączyła równolegle porty. Po pocięciu blaszki i zaizolowaniu końców należało nanieść cynę. Przy temperaturze 400°C i topnikowi cyna chwyciła bez problemów. Elementy te zostały przyklejone klejem na gorąco. Następnie należało uciąć odpowiednie kawałki kabli, aby pasowały na długość. Przedstawia to rysunek 6.9. Całość natomiast po podłączeniu prezentuje się jak na rysunku 6.10.



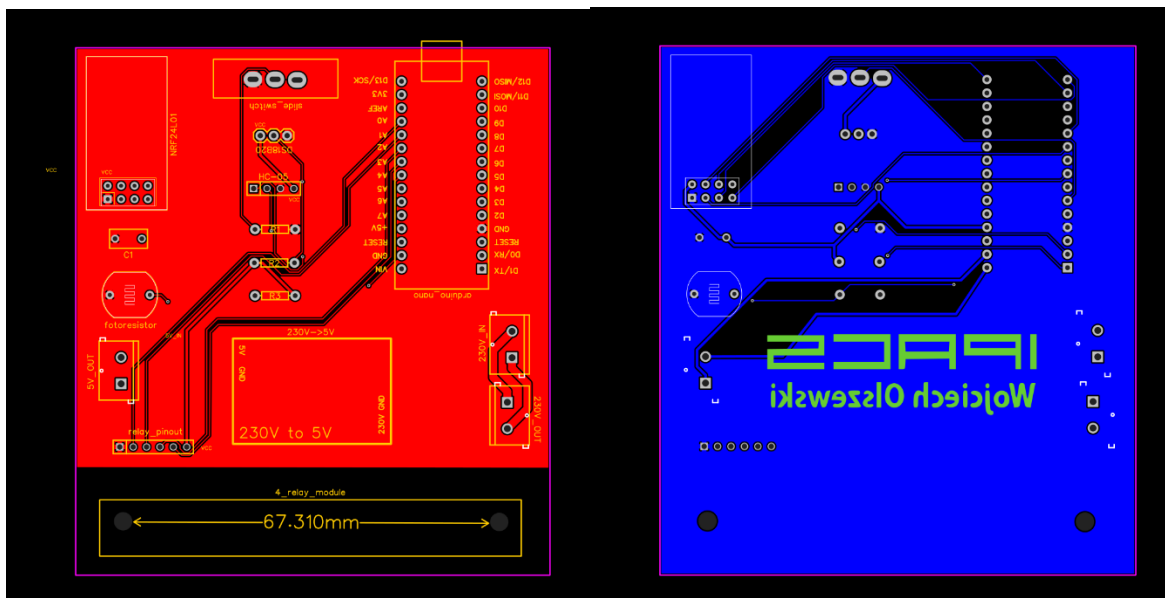
Rys. 6.9 Wnętrze zmodyfikowanej listwy



Rys. 6.10 Drugi prototyp listwy

Etap 9

Po dłuższych testach okazało się, że istnieją występujące losowo problemy z transmisją. Powodem jest niska moc nadajników. W tym miejscu wprowadzono rozwiązania opisane wcześniej, aby zapewnić działanie systemu w trybie ciągłym, bez przerw. Zaprojektowano także nową dwuwarstwową płytkę drukowaną prototypową, która miała być wykonana przez fabrykę. Projekt PCB wraz z polygonem przedstawia rysunek 6.11. Na tym etapie zakończono pracę sprzętową oraz programistyczną i zamknięto projekt.



Rys. 6.11 Trzeci prototyp listwy

6.5. Testowanie systemu

Testy możemy podzielić na dwa rodzaje ze względu na miejsce ich odbycia:

- Testy w środowisku akademickim
- Testy w domu jednorodzinnym

oraz ze względu na ich naturę:

- Testy oprogramowania
- Testy transmisji radiowej

Badania przeprowadzane w domach studenckich polegały raczej na bieżących eksperymentach pracy sieci niż efektywnej transmisji. Należy wziąć jednak pod uwagę, że ściany nośne budynku są na tyle grube, aby skutecznie tłumić odgłosy zza ściany. Nie można już tego powiedzieć o stropie.

W domu jednorodzinnym badania polegały tylko i wyłącznie na sprawdzeniu zasięgu, gdyż nie można tam było pracować nad oprogramowaniem z powodów braku odpowiedniego sprzętu i stanowiska.

Transmisja w akademiku była zależna od poziomów mocy nadajników, przy wyborze mocy HIGH, bo z MAX (czyli 0dB) wydawało się, że NRF sobie nie radził, zasięg można było przyrównać do zasięgu średniego mieszkania. Środowisko testowe nie było przyjazne, ponieważ mówimy tutaj o grubych ścianach nośnych oraz dużej ilości drzwi i metalowych

elementów na korytarzach. W linii prostej transmisja na odległości 25m w korytarzu przebiegała bez zarzutu. Jeśli chodzi o pokoje, to pomiędzy segmentami transmisja ta także działała, jednak nie można powiedzieć, że była dobra.

W domku rodzinnym, który posiada 2 piętra, nie osiągnięto całkowitego pokrycia. Winna jest mała moc wypromieniowywana przez anteny o małym zysku. Zasięgiem była objęta część pierwszego piętra jak i pokój nad nadajnikiem, a także część górnego piętra. Pokrycie można poprawić zmieniając ustawienie listw. Musimy pamiętać, że będą ona leżeć gdzieś na ziemi, być może za jakąś szafą. Każda przeszkoda wnosi jakieś tłumienie, ale i możliwość większego zasięgu, z powodu retrakcji. Fala trafiająca do listwy z odbicia może być mocniej wytłumiona niż fala, która została wytłumiona poprzez zagięcie na przeszkodzie i dojście kierunkowe wprost do listwy. Dużo fal odbitych i bezpośrednich wzajemnie się niweluje przez dojście w różnych fazach. Dodatkowo dochodzą echa sygnału z opóźnieniem. Wniosek jest taki, że w środowisku domowym, przeszkody mogą wpłynąć tak negatywnie jak i pozytywnie na zasięg.

Efekt nasilonych odbić oraz wygaszania się fal najwidoczniejszy był w obrębie jednego pokoju, gdzie moduły były w odległości mniej niż 3 metry i należało się przed tym uchronić możliwością sterowania mocą, jak i weryfikacją programową odbieranych ech.

Dzięki trybowi debugowania, programowi *gdb* oraz logami na stronie można było łatwo wyłapywać wszelkie problemy w działaniu systemu. Przykładowo, jeśli autor wiedział, że przez jakiś czas system nie będzie modyfikowany, to mógł w trybie graficznym uruchomić program z logami w trybie *fast* i po jakimś czasie, jeśli program nie działał poprawnie, to dokładnie wiadomo było, w jakim miejscu i dlaczego.

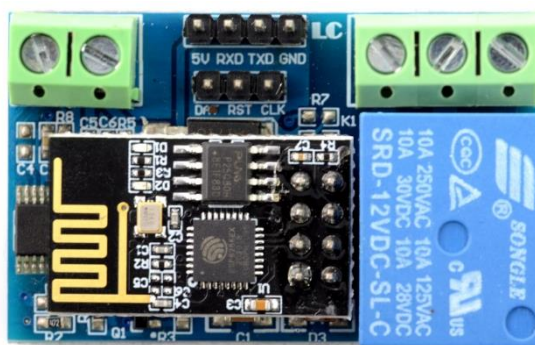
Dość częstym scenariuszem było zerwanie komunikacji międzypokojowej, gdy w pokoju ktoś przebywał. Nie wyjaśnia to niestety, dlaczego listwa pakiety odbierała i nadawała zwrotnie, a stacja matka nadanych już nie odbierała. Zmiana mocy nic w tej kwestii nie zmieniała. Nadajniki potrafiły także w ciągu nocy zerwać połączenie na parę minut. Określa to słabość nadajników NRF24, czyli wrażliwość na przeszkody, a wynika to z małej mocy i słabej modulacji. Nie można jednak mieć wszystkiego: jeśli prostota oraz niska cena znajdują się po jednej stronie wymagań projektowych, to niska moc i mały zasięg lokują się po przeciwnej stronie.

7. Przegląd podobnych rozwiązań

Autor postanowił zbudować od podstaw projekt mimo istnienia podobnych rozwiązań w tym dostępnych w sprzedaży. Wynika to z chęci zdobycia doświadczenia i stworzenia rozwiązania funkcjonującego nieco inaczej od już istniejących, według własnych założeń. Stworzyło to okazję do przypomnienia sobie wielu wiadomości oraz samorozwoju. Dostępne układy nie posiadają na pokładzie zintegrowanych czujników i jeśliby trzeba było zamknąć cały układ pod obudową listwy, stanowiłoby to problem, z racji z góry zdefiniowanych rozmiarów. Autor może bez problemów dobrać wielkość i kształt układu do potrzeb. W prototypie projektu został wybrany moduł z 4 przekaźnikami, ale nic nie stoi na przeszkodzie, aby wykorzystać pojedyncze przekaźniki z optoizolatorami.

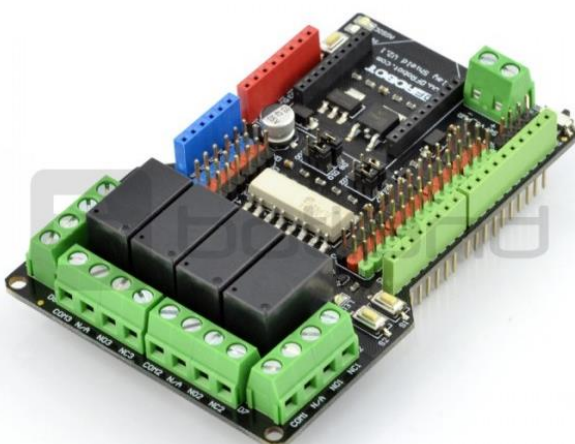
Poniżej scharakteryzowano rozwiązania dostępne dla przykładu w sklepie *Botland*.

- *Moduł WiFi ESP8266 + przekaźnik - styki 10A/250VAC - cewka 12V [24].*
Jest to moduł (rys. 7.1), który posiada w gnieździe goldpin miejsce na moduł *ESP8266* pozwalający na pracę w 3 trybach komunikacyjnych: *Station* (Klient), *AP* (Serwer) oraz *Station + AP*. Pracuje on na bazie mikroprocesora *Tensilica L106 32-bit*. Można go podłączyć do np. *Arduino* poprzez *UART* i w ten sposób sterować.



Rys. 7.1 Moduł WiFi ESP8266 + przekaźnik

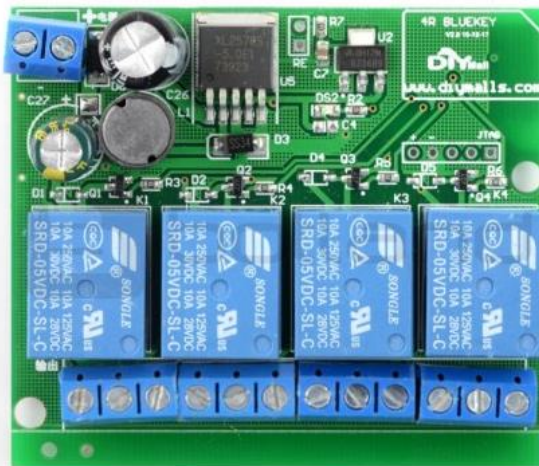
- *DFRobot Relay Shield - przekaźniki dla Arduino v2.1 [25].*
Jest to moduł (rys. 7.2) przygotowany do pracy z *Arduino* oraz modułem komunikacji bezprzewodowej *XBee*.



Rys. 7.2 DFRobot Relay Shield

- *Moduł przekaźników 4 kanały + Bluetooth 4.0 BLE - styki 10A/250V - cewka 5V [26].*

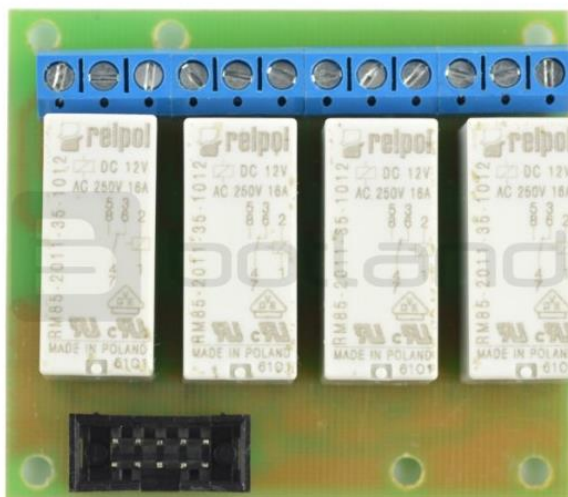
Jest to moduł (rys. 7.3) w pełni autonomiczny, sterowany przez komendy wysyłane przez Bluetooth. Niestety standard Bluetooth ma niski zasięg, z racji zaawansowanych protokołów ochrony transmisji.



Rys. 7.3 Moduł przekaźników 4 kanały + Bluetooth [26]

- *Tinycontrol GSMKON-010 - płytka przekaźników 4x 16A / cewka 12V do GSM/LAN kontrolera [27].*

Jest to moduł (rys. 7.4), który jest przygotowany do pracy z kontrolerami LAN (przewodowa transmisja) lub GSM (transmisja bezprzewodowa ogólnosiwiatowa). Dodatkowo ma lepszą wytrzymałość samych przekaźników.



Rys. 7.4 Tinycontrol GSMKON-010 - płytka przekaźników [27]

Istnieje wiele różnych rozwiązań, przedstawione tutaj dostępne są w najpopularniejszym polskim internetowym sklepie z elektroniką. Mają one swoje wady i zalety. Najbliższym zaprojektowanemu systemowi jest pierwszy przykład z modułami Wi-Fi ESP.

Jeśliby zamienić moduły NRF24 na ESPXX, powiększyło by to zasięg, kontrola mocy systemu byłaby lepsza, znalazłbyśmy też straty sygnału, a także algorytm transmisyjny byłby inteligentniejszy.

Musimy odpowiedzieć sobie na pytanie, czy warto przyjmować najłatwiejsze rozwiązania? Praca w trudniejszym środowisku, przy gorszych parametrach i sprzęcie nauczy nas dużo więcej niżli gotowe, bardzo dobrze działające rozwiązanie. Gdyby projekt polegał na przesyłaniu ważnych danych i transmisja musiałaby być stabilna, wtedy możemy iść na ustępstwa. W podjętym w pracy projekcie czekając na odpowiedź czujnika temperatury godzimy się na wolną transmisję, która, z racji małej liczby pakietów na sekundę (aż trzy), może być (i to bardzo) słaba. Tworząc własne rozwiązanie protokołu transmisji nauczymy się, jak zarządzać programowo połączeniem oraz jak uchronić się przed pewnymi niekorzystnymi scenariuszami. Pewne rzeczy można zautomatyzować, inne natomiast nie. Mając zapas czasu i moduły, które są dość tanie i łatwe w obsłudze, można sporo osiągnąć i wykorzystać jak najlepiej sprzęt.

Omawiany w pracy projekt bazuje na nadajnikach, które nie są obecne w żadnym znalezionym urządzeniu. Autor zwrócił na to uwagę i było to jednym z powodów ich wyboru. Dodatkowo system posiada czujniki, które mogą być dowolnie wyłączane jak i włączane oraz być całkowicie wypięte z układu. Zastosowano też przełącznik przesuwany, którym można załączać przekaźniki niezależnie od stanu komunikacji. Układ można dowolnie przeprogramować, tak aby działał w innych trybach. ESP8266 komunikuje się poprzez SPI i można, dla przykładu z użyciem przejściówki (wyjścia nie są w tej samej kolejności), podłączyć do istniejącej płytki drukowanej moduł Wi-Fi. Stwarza to dalsze możliwości rozwoju projektu.

8. Podsumowanie

Naczelnym założeniem projektu było jego funkcjonowanie bezprzewodowe oraz możliwość automatyzacji zleconych systemowi zadań. Założenia udało się spełnić. W trakcie kreowania końcowej wersji, udało się dokonać paru modyfikacji udoskonalających projekt.

Projekt został wykonany z zapasem czasu, pozwoliło to na głębszą analizę jego działania, w celu odnalezienia błędów projektowych oraz programistycznych. Interfejs systemu był starannie tworzony i poddawany krytyce osób trzecich, aby był on przejrzysty i czytelny dla każdego, nie tylko dla autora.

Testy wykazały jasno, że do tego projektu moduły NRF24L01 się nie nadają, gdyż nie pokrywają zasięgiem mieszkania, nie wspominając o domu jednorodzinnym. Jeśli weźmiemy pod uwagę połączenie typu „w obrębie pokoju” lub „pokój-pokój obok (nad)” to system powinien funkcjonować. Według autora, parametry nadajników wykorzystano jak najlepiej, by zagwarantować dobrą transmisję, niestety działają one w sposób mało kontrolowany, pomimo wielu starań.

Sieć ostatecznie działa bez zarzutu w małym otoczeniu. Można się domyślać, że gdyby sterować nią na podwórku, np. do załączania świateł itp. działała ona by lepiej niż w środowisku mieszkaniowym. Rozwiązaniem, które mogłoby polepszyć zasięg w obrębie domu (i nie tylko) mogłoby być stworzenie przekaźników, które przekazywały by dalej odebraną wiadomość. W skrajnym przypadku taką rolę można także przypisać samym listwom, należałoby wtedy zainteresować się tematem kolizji i wpływem nadmiaru ruchu na pracę zaprojektowanej sieci. Nic nie stoi na przeszkodzie, aby takie zastosowanie wprowadzić w życie w kolejnym etapie rozwoju projektu.

Dość problematyczny jest czas konwersji danych temperaturowych z czujnika DS18B20, który trwa 180ms. Gdyby nie to, można by było przyspieszyć odpytywanie listw i sama transmisja uzyskałaby większą niezawodność. Rozwiązaniem takiego problemu mogłoby się stać użycie np. mikrokontroler Attiny do obsługi czujnika i Atmega jako sterownika, który działałby niezależnie od danych z czujnika.

Wiedząc już jak funkcjonuje zaprojektowany system można się zastanawiać nad jego szerszym zastosowaniem. Opracowane rozwiązanie nie dotyczy tylko sterowania gniazdami 230V. Przełączniki są galwanicznie odizolowane od zasilania swojego układu, a także od całego wytworzonego modułu, toteż mogą służyć do sterowania dowolnymi sygnałami i załączania ich w odpowiedniej sekwencji. Modułu nie trzeba montować przy listwie lub w niej. bazując na gotowym projekcie, można zaprojektować inny kształt płytki drukowanej i dołączyć inne złącza. Słowo listwa, w pracy używane także jako adapter, nie musi oznaczać listwy zasilającej, a może oznaczać panel wejść/wyjść, który w szczególnym przypadku (przedłużacz) stanie się listwą zasilającą.

System mimo wszystko przerósł oczekiwania autora i przynosi mu dumę, z powodów czysto prozaicznych. Wykorzystane i odświeżone zostały umiejętności, które nie są łatwe w opanowaniu i są kwintesencją lat pracy na studiach oraz hobbystycznych. Zmierzenie się

z opisanymi w pracy problemami pobudziło myślenie analityczne projektanta i zrodziło działające i rozwojowe rozwiązania.

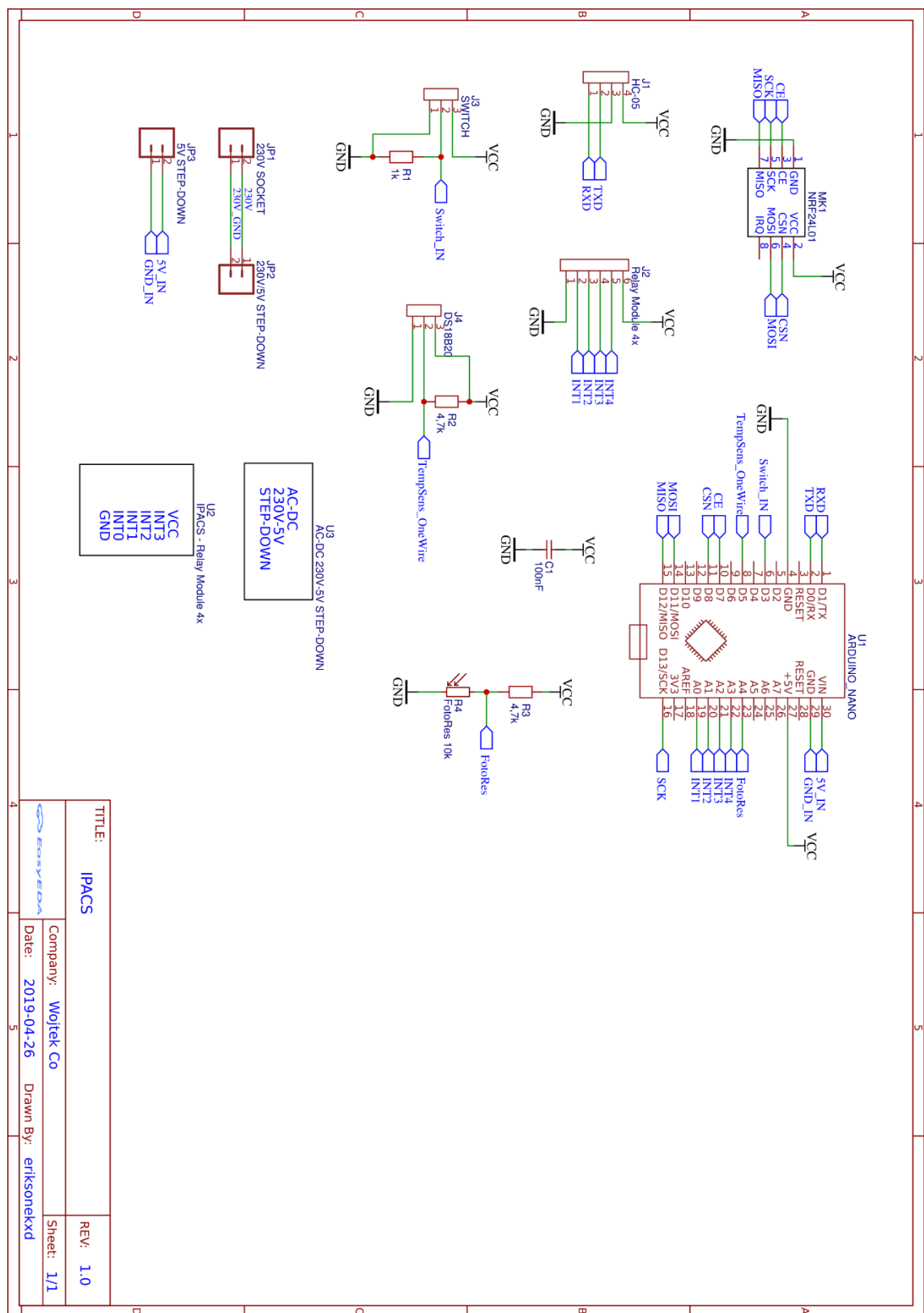
Bibliografia

Wykaz w kolejności występowania odwołań w tekście odwołania:

- [1] Donald Norris “Raspberry Pi Niesamowite Projekty” Wyd. Helion 2014
- [2] Raspberry Pi 3 model B
<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>
- [3] Arduino Older Boards <https://www.arduino.cc/en/Main/Boards>
- [4] “Arduino 65 Praktycznych Projektów” John Boxal, Wyd. Helion 2014
- [5] Arduino Nano Tech Specs <https://store.arduino.cc/arduino-nano>
- [6] DS18B20 <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>
- [7] Fotorezystor 20K Ω
https://reference.digilentinc.com/_media/ni:photoresistor_ds.pdf
- [8] NRF24L01
https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf
- [9] About MariaDB <https://mariadb.org/about/>
- [10] Wprowadzenie do systemu MySQL
http://informatyka.2ap.pl/ftp/technik_inf/kurs_mysql.pdf
- [11] Encja (bazy danych) [https://pl.wikipedia.org/wiki/Encja_\(bazy_danych\)](https://pl.wikipedia.org/wiki/Encja_(bazy_danych))
- [12] Rekord (informatyka)
[https://pl.wikipedia.org/wiki/Rekord_\(informatyka\)#Bazy_danych](https://pl.wikipedia.org/wiki/Rekord_(informatyka)#Bazy_danych)
- [13] Klucz główny [https://pl.wikipedia.org/wiki/Klucz_główny](https://pl.wikipedia.org/wiki/Klucz_g%C5%82%C3%B3wny)
- [14] Klasa NRF24 <https://tmrh20.github.io/RF24/classRF24.html>
- [15] Definition of: ISM band
<https://www.pcmag.com/encyclopedia/term/45467/ism-band>
- [16] Sesje <https://phpkurs.pl/obsługa-sesji/>
- [17] Czym są pliki cookies w przeglądarce?
<https://pomoc.home.pl/baza-wiedzy/czym-wlasciwie-sa-pliki-cookies-ciasteczka-w-przegladarce>
- [18] Metatag https://www.w3schools.com/tags/tag_meta.asp
- [19] Setting up an Apache Web Server on a Raspberry Pi
<https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md>
- [20] Tutorial – Install MySQL server on Raspberry Pi
<https://www.stewright.me/2014/06/tutorial-install-mysql-server-on-raspberry-pi/>

- [21] MySQL++ <https://tangentsoft.com/mysqlpp/home>
- [22] Optimized High Speed NRF24L01+ Driver Class Documentation
<https://tmrh20.github.io/RF24/index.html>
- [23] A beginners Guide to Cron Jobs
<https://www.ostechnix.com/a-beginners-guide-to-cron-jobs/>
- [24] Moduł WiFi ESP8266 + przekaźnik - styki 10A/250VAC - cewka 12V
https://botland.com.pl/pl/nawosci/9391-modul-wifi-esp8266-przekaznik-styki-10a250vac-cewka-12v.html?search_query=przekaznik+esp&results=5
- [25] DFRobot Relay Shield - przekaźniki dla Arduino v2.1
https://botland.com.pl/pl/arduino-shield-ekspandery-wyprowadzen/2731-dfrobot-relay-shield-przekazniki-dla-arduino-v21.html?search_query=przekaznik&results=190
- [26] Moduł przekaźników 4 kanały + Bluetooth 4.0 BLE - styki 10A/250V - cewka 5V
https://botland.com.pl/pl/moduly-przekaznikow/10909-modul-przekaznikow-4-kanały-bluetooth-40-ble-styki-10a250v-cewka-5v.html?search_query=przekaznik&results=190
- [27] Tinycontrol GSMKON-010 - płytki przekaźników 4x 16A / cewka 12V do GSM/LAN kontrolera
https://botland.com.pl/pl/kontrolery-tinycontrol/8291-tinycontrol-gsmkon-010-plytka-przekaznikow-4x-16a-cewka-12v-do-gsmlan-kontrolera.html?search_query=przekaznik&results=190

Dodatek A Schemat prototypu listwy



TITLE: IPACS

REV: 1.0

Company: Wojtek Co

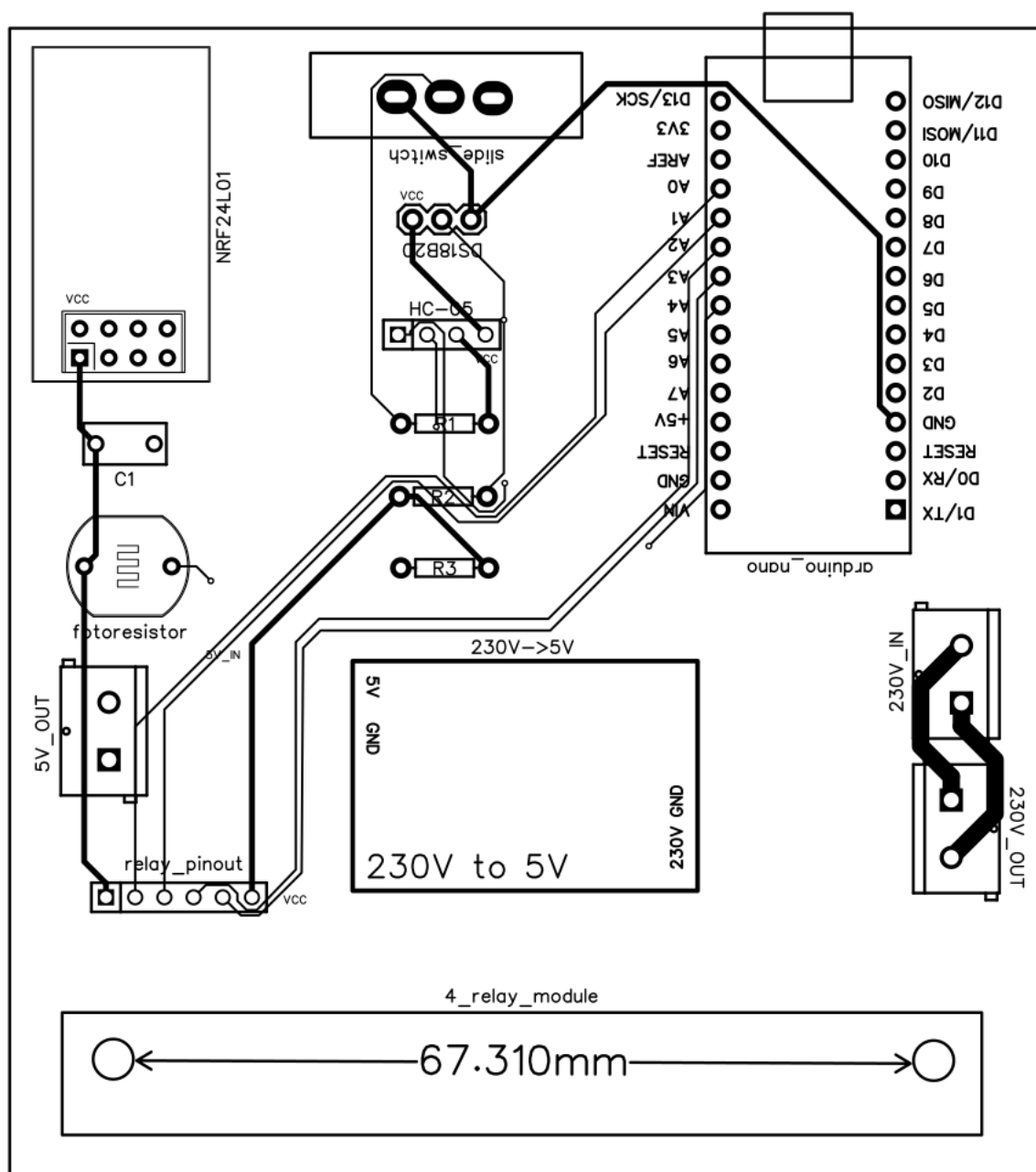
Date: 2019-04-26

Drawn By: eriksonekxd

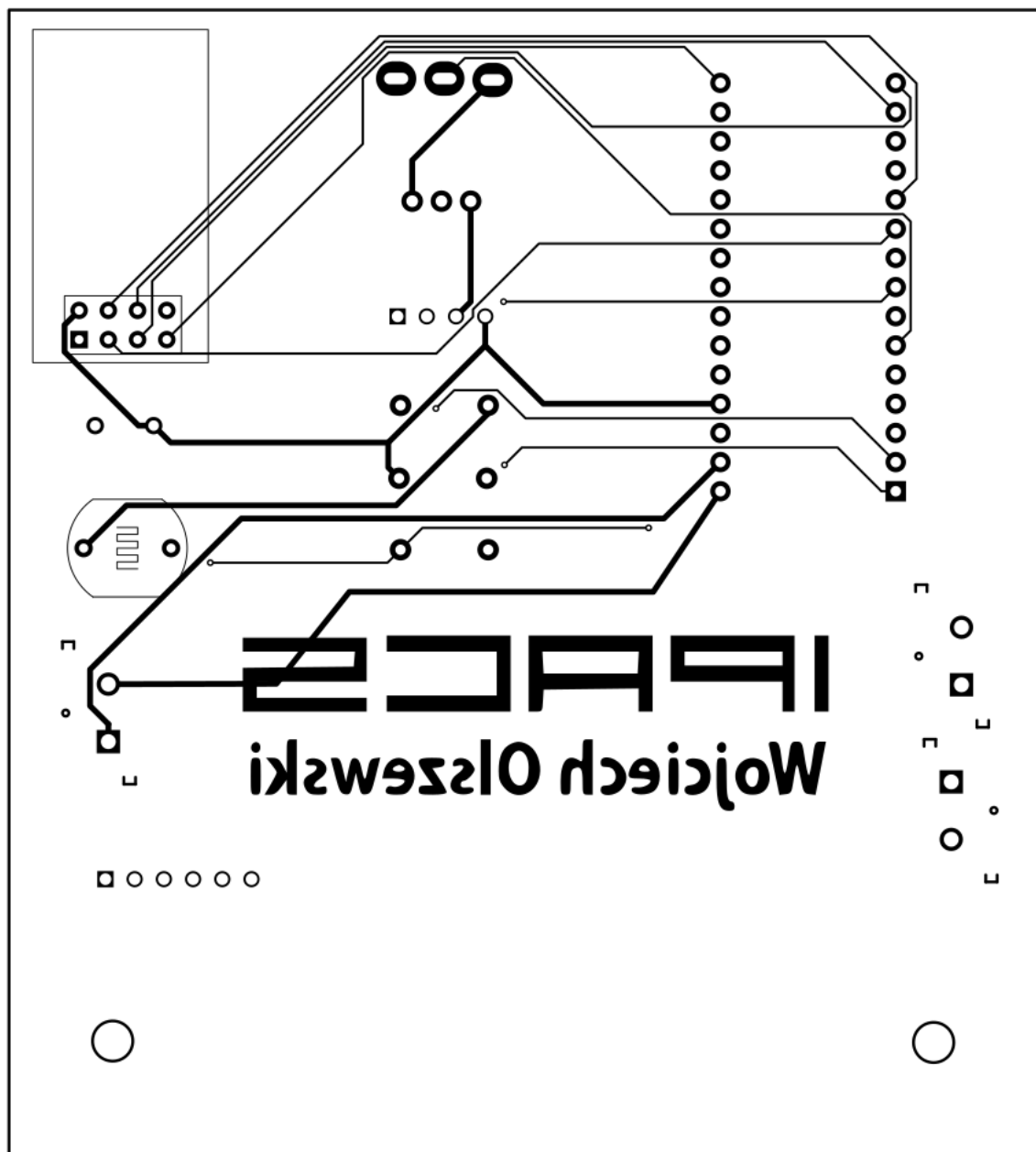
Sheet: 1/1

Dodatek B PCB prototypu układu

Pierwsza warstwa



Druga warstwa



Dodatek C Fragment noty katalogowej NRF24L01



nRF24L01+

Single Chip 2.4GHz Transceiver

Preliminary Product Specification v1.0

Key Features

- Worldwide 2.4GHz ISM band operation
- 250kbps, 1Mbps and 2Mbps on air data rates
- Ultra low power operation
- 11.3mA TX at 0dBm output power
- 13.5mA RX at 2Mbps air data rate
- 900nA in power down
- 26µA in standby-I
- On chip voltage regulator
- 1.9 to 3.6V supply range
- Enhanced ShockBurst™
- Automatic packet handling
- Auto packet transaction handling
- 6 data pipe MultiCeiver™
- Drop-in compatibility with nRF24L01
- On-air compatible in 250kbps and 1Mbps with nRF2401A, nRF2402, nRF24E1 and nRF24E2
- Low cost BOM
- ±60ppm 16MHz crystal
- 5V tolerant inputs
- Compact 20-pin 4x4mm QFN package

Applications

- Wireless PC Peripherals
- Mouse, keyboards and remotes
- 3-in-1 desktop bundles
- Advanced Media center remote controls
- VoIP headsets
- Game controllers
- Sports watches and sensors
- RF remote controls for consumer electronics
- Home and commercial automation
- Ultra low power sensor networks
- Active RFID
- Asset tracking systems
- Toys

All rights reserved.

Reproduction in whole or in part is prohibited without the prior written permission of the copyright holder.
March 2008

Dodatek D Wykaz elementów

Elementy listwy:

- Moduł radiowy NRF24L01
- Platforma Arduino Nano
- Moduł przekaźników
- Przełącznik przesuwany
- Czujnik temperatury DS18B20
- Rezystor 4,7k x3
- Fotorezystor 10k
- Przetwornica 230V/5V STEP-DOWN
- Kondensator 100nF
- Złącze ARK x3
- Listwy goldpin x6

Elementy stacji matki:

- Mikrokomputer Raspberry Pi
- Moduł radiowy NRF24L01

