

Implementacja systemu zarządzania nadwyżką produkcji energii elektrycznej z instalacji fotowoltaicznej

(Implementation of the system for managing surplus electricity production
from a photovoltaic installation)

Krzysztof Juszczyk

Praca inżynierska

Promotor: dr Marek Materzok

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

31 stycznia 2022

Streszczenie

Obecnie występuje wzmożona tendencja na inwestowanie w instalacje fotowoltaiczne. Najkorzystniejsze warunki rozliczania prosumentów proponowane przez dostawców energii to zerowy rachunek za prąd w danym okresie rozliczeniowym. W sytuacji, gdy doświadczamy szczególnie słonecznej pogody, nadwyżka wygenerowanego prądu bezpowrotnie przepada na korzyść dostawcy. Możemy zatem pochylić się ku wykorzystaniu kolejnej technologii zyskującej na popularności – blockchain i zjawiska „wydobycia” kryptowalut metodą Proof of Work. System ma za zadanie na bieżąco monitorować bilans zużycia prądu w gospodarstwie domowym, aby w czasie rzeczywistym zarządzać zasobami sprzętowymi, które generują uzyskanie cyfrowego przychodu i maksymalizują czerpanie zysku z wyprodukowanej energii słonecznej.

There is currently an increased tendency to invest in photovoltaic installations. The most favourable billing conditions for prosumers offered by energy providers are zero electricity bill per billing period. In this situation, when we experience particularly sunny weather, the surplus electricity generated is irretrievably lost to the energy provider. We can therefore lean towards using another technology gaining popularity, blockchain and the phenomenon of "mining" cryptocurrencies using the Proof of Work method. The system is designed to continuously monitor the balance of a household's electricity consumption in order to manage, in real time, the hardware resources that generate digital revenue and maximise profit from the solar energy produced.

Spis treści

Wstęp	7
1. Wymagania projektu	9
1.1. Specyfika działania instalacji fotowoltaicznej	9
1.2. Zdefiniowanie wymagań projektu	9
2. Realizacja wymagań	11
2.1. Protokół komunikacji	11
2.2. Monitorowanie bilansu zużycia/produkcji prądu	12
2.3. Monitorowanie zużycia prądu przez koparki	13
2.4. Kontrolowanie pracy koparek	14
2.5. Gromadzenie pomiarów danych prądowych	16
2.5.1. Dane odczytywane w rozdzielnicy prądowej	16
2.5.2. Ogólne dane zużycia energii przez koparki	17
2.5.3. Zużycie energii uzyskanej z sieci energetycznej przez koparki .	17
2.6. Centralny serwer projektu	17
2.7. Tunelowanie VPN	18
3. Implementacja rozwiązań	19
3.1. Implementacja - Guard	19
3.1.1. Część sprzętowa	19
3.1.2. Część programowa	21
3.1.3. Tematy MQTT obsługiwane przez Guard'a	23
3.2. Implementacja - główny serwer systemu Mithra	26

3.2.1.	Uruchomienie programu	27
3.2.2.	Konfiguracja systemu	27
3.2.3.	Etap wstępny działania programu	29
3.2.4.	Podział zadań pomiędzy wątki	30
3.2.5.	Działanie wątków odczytujących wiadomości MQTT	32
3.2.6.	Wątek obsługujący komunikację z bazą danych	32
3.2.7.	Zadania głównego wątku	32
3.2.8.	Wykluczanie koparek	33
3.2.9.	Algorytm planowania zużycia zasobów energii	34
4.	Uruchomienie systemu	39
4.1.	Serwer PostgreSQL i borker MQTT	39
4.2.	Urządzenia Shelly	39
4.3.	Wgrywanie programu Guarda	40
4.4.	Kompilacja i uruchomienie Mithry	40
	Bibliografia	41

Wstęp

W czasie wzmożonego zainteresowania technologiami ekologicznymi, dosyć popularnym zjawiskiem stało się produkowanie energii elektrycznej przez gospodarstwa domowe, za pomocą instalacji fotowoltaicznych, w celu obniżenia rachunku za prąd. Dostawcy energii zazwyczaj proponują prosumentom następujące warunki wykorzystywania fotowoltaiki: w danym okresie rozliczeniowym (jest to zazwyczaj rok) prosument produkuje energię, którą na bieżąco zużywa lub jej nadmiar oddaje do sieci energetycznej. Jeśli zapotrzebowanie gospodarstwa na prąd jest większe niż jego produkcja, to pobiera się energię z sieci. Pod koniec okresu rozliczeniowego następuje podsumowanie. Jeśli prosument:

- wyprodukował i oddał więcej energii, niż pobrał z sieci energetycznej, to nie płaci rachunku za prąd,
- zużył więcej energii, niż wyprodukował, to zostaje opłacona tylko różnica energii pobranej od energii wyprodukowanej.

Łatwo zauważyć, że taki układ jest korzystniejszy dla dostawców prądu, czyli za nadmiar oddanej energii do sieci prosument nie otrzymuje żadnego wynagrodzenia. Należy jeszcze wspomnieć, że zazwyczaj dostawcy zastrzegają sobie, że tylko część oddanej energii może zostać z powrotem wykorzystana (np. 80%) w ramach opłat przesyłowych i utrzymania sieci energetycznej.

Nierealne jest przewidzenie warunków atmosferycznych i dokładnego zapotrzebowania gospodarstwa na prąd w ciągu okresu rozliczeniowego, więc prosument, przed przystąpieniem do inwestycji, musi oszacować, o jakiej mocy znamionowej dobrać instalację fotowoltaiczną, aby w każdym okresie rozliczeniowym realizować najbardziej optymistyczny scenariusz. Dlatego w sytuacji, gdy doświadczamy wyjątkowo słonecznej pogody, a produkcja prądu przewyższa nasze potrzeby, możemy się pokusić o zagospodarowanie nadmiarowych zasobów energii. Tutaj przychodzi na pomoc technologia blockchain i zyskujące na popularności kryptowaluty. Zjawisko wydobywania cyfrowych aktywów metodą *Proof of Work* może posłużyć do zużycia nadmiarowych zasobów energii i przeobrażenia ich w cyfrową formę przychodu, jednocześnie przyczyniając się do przyspieszenia tempa zwrotu inwestycji w fotowoltaikę.

Celem tej pracy jest stworzenie samodzielnego systemu, który monitoruje parametry produkcji/zużycia prądu i stara się w czasie rzeczywistym wykorzystać dostępne zasoby energetyczne i sprzętowe (tzw. koparki), do wydobywania kryptowaluty. W pierwszym rozdziale zostanie przedstawiony szereg wymagań do spełnienia, aby centralny serwer systemu uzyskał pełnię możliwości funkcjonowania, był w stanie monitorować bilans zużycia prądu przez gospodarstwo oraz sterował działaniem koparek.

Rozdział 1.

Wymagania projektu

1.1. Specyfika działania instalacji fotowoltaicznej

Zanim zostaną przedstawione poszczególne wymagania systemu, należy przedstawić specyfikę działania instalacji fotowoltaicznej, do której będzie dostosowany system. Wygląda to w sposób następujący: wyprodukowany prąd zasila domową trójfazową instalację elektryczną, jeśli sumaryczna moc urządzeń zasilanych w ramach jednej fazy nie przekracza mocy produkowanego prądu, to nadwyżka energii zostaje przekazana do publicznej sieci energetycznej. Jak wcześniej wspomniano, część energii zwróconej do sieci może zostać spożytkowana później za darmo, przed końcem okresu rozliczeniowego. Wartym uzmysłowienia jest fakt, że energia oddana w ramach jednej fazy może zostać skonsumowana w obrębie innej fazy.

1.2. Zdefiniowanie wymagań projektu

W celu realizacji tego projektu należy pochylić się nad spełnieniem następujących wymagań:

- monitorowanie parametrów prądowych w rozdzielniczy elektrycznej,
- monitorowanie zużycia prądu przez koparki,
- kontrolowanie pracy koparek,
- komunikacja urządzeń pomiarowych i wykonawczych systemu z centralnym serwerem projektu,
- centralny serwer projektu,
- gromadzenie i przechowywanie historycznych danych prądowych systemu,
- udostępnienie interfejsu sieciowego osiągalnego z sieci Internet, umożliwiającego dostęp do lokalnych urządzeń sieciowych.

Pierwsze dwa wymagania zostaną zrealizowane za pomocą gotowych rozwiązań dla automatyki domowej dostępnej na rynku. Ostatni wymóg jest ponadplanowy i został spełniony w celu ułatwienia autorowi pracy rozwijania projektu na odległość.

Rozdział 2.

Realizacja wymagań

2.1. Protokół komunikacji

Przed omówieniem poszczególnych urządzeń wchodzących w skład systemu, należy przyrzeć się zastosowanej metodzie komunikacji w systemie. Popularnym protokołem wymiany informacji spotykanym w automatyce domowej opartej o model TCP/IP jest MQTT [1]. Centralnym punktem komunikacji jest broker MQTT, czyli serwer, z którym łączą się klienci i za jego pośrednictwem komunikują się ze sobą. Wymiana danych pomiędzy klientami opiera się na dwóch czynnościach:

- subskrypcji jednego lub wielu tematów,
- publikowaniu wiadomości pod zadany temat.

Jeśli klient opublikuje zestaw informacji pod konkretnym tematem, to każdy klient, który subskrybował ten temat wcześniej, otrzyma kopię informacji. Tematy mogą być dowolnie tworzone przez klientów. Broker dodatkowo implementuje funkcjonalność uwierzytelniania klientów loginem i hasłem, co pozwala zwiększyć poziom bezpieczeństwa i chroni przed ingerencją niechcianych użytkowników w proces komunikacji systemu.

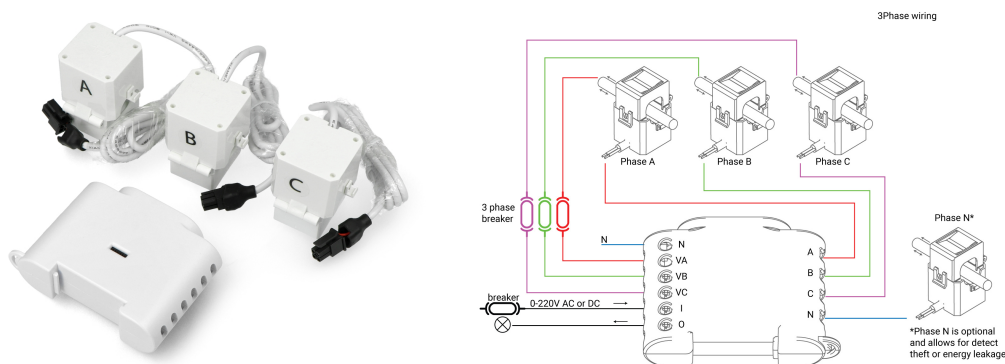
Implementacja brokera wykorzystana w projekcie to Mosquitto [2] dostępna w systemach operacyjnych opartych na jądrze Linux. Wraz z implementacją serwera MQTT dostarczony jest zestaw narzędzi dostępnych z poziomu wiersza poleceń pozwalających na komunikację z brokerem Mosquitto. Tymi narzędziami są programy:

- **mosquitto_pub** [3] - pozwala na wysłanie do brokera wiadomości pod zadany temat.
- **mosquitto_sub** [4] - odbiera i wypisuje na standardowe wyjście wszystkie wiadomości z subskrybowanych tematów, podanych w opcjach programu.

Powyższe narzędzia niejednokrotnie służyły do rozwijania, testowania i monitorowania działania implementacji całego projektu.

2.2. Monitorowanie bilansu zużycia/produkcji prądu

W celu monitorowania stanu energetycznego gospodarstwa zostało wykorzystane rozwiązanie oferowane przez firmę Shelly. Urządnie Shelly 3EM [5] umożliwia pomiar energii każdej z trzech faz, z uwzględnieniem czy prąd jest pobierany, czy oddawany do sieci. Urządzenie posiada moduł WiFi, dzięki czemu pozwala na zdalny podgląd parametrów zarówno z poziomu przeglądarki internetowej, jak i aplikacji na smartfony. Zapewnia również wsparcie dla protokołu MQTT i pod wyznaczonymi tematami publikuje cyklicznie parametry prądowe.



Rysunek 2.1: Zdjęcie poglądowe i schemat podłączenia urządzenia

Specyfikację wszystkich możliwych wartości, jakie rejestruje urządzenie można znaleźć na stronie producenta [6]. Zostaną tutaj przedstawione i opisane wyłącznie parametry wykorzystane w projekcie.

Wszystkie wymienione poniżej parametry prądowe są dostępne za pośrednictwem protokołu MQTT pod tematami:

`shellies/shellyem3-<device_id>/emeter/<i>/<suffix>`,

gdzie `<device_id>` oznacza numer identyfikacyjny urządzenia, `<i>` fazę, na której odbył się pomiar wartości ($i \in \{0, 1, 2\}$), natomiast zmienna `<suffix>` mierzony parametr:

- `energy` – licznik energii pobranej z sieci energetycznej przez ostatnią minutę, wyrażona w jednostce wato-minuty,
- `returned_energy` – licznik energii oddanej do sieci energetycznej przez ostatnią minutę, wyrażona w jednostce wato-minuty,

- **total** – całkowita energia pobrana z sieci energetycznej, wyrażona w wato-godzinach,
- **total_returned** – całkowita energia oddana do sieci energetycznej, wyrażona w wato-godzinach.

Należy zaznaczyć, że wartości **total** i **total_returned** są przechowywane w pamięci nieulotnej urządzenia i nie są narażone na utratę w przypadku utracenia zasilania lub łączności WiFi. Te parametry mogą zostać ręcznie wyzerowane. Dodatkowo, gdy pod tematem **shellies/command** zostanie opublikowana wiadomość o treści **announce**, urządzenie opublikuje wiadomość w formacie JSON, której treść będzie zawierała informacje ogólne o urządzeniu.

2.3. Monitorowanie zużycia prądu przez koparki

Zadanie monitorowania zużycia prądu przez koparki również zostanie powierzone układowi stworzonemu przez firmę Shelly. Urządzenie Shelly PlugS [7] jest jednofazowym miernikiem zużycia energii elektrycznej i podobnie jak poprzednie urządzenie jest wyposażone w moduł WiFi i wspiera protokół MQTT.



Rysunek 2.2: Zdjęcie poglądowe urządzenia Shelly PlugS

Odczyt parametrów jest dostępny pod tematem: **shellies/shellyplug-s-<device_id>/relay/0/<suffix>**, gdzie **<device_id>** oznacza numer identyfikacyjny urządzenia, a **<suffix>** mierzony parametr. Monitorowane parametry w projekcie to:

- **power** – reprezentuje chwilową moc, jaką odznacza się urządzenie podłączone do miernika, wyrażona w watach,

- **energy** – reprezentuje ilość energii pobranej przez urządzenie od momentu włączenia zasilania w mierniku, wyrażona w wato-minutach.

Miernik dodatkowo posiada funkcjonalność fizycznego włączania i wyłączenia zasilania urządzenia, które zostało do niego podłączone. By to zrobić, należy pod tematem `shellies/shellyplug-s-<device_id>/relay/0/command` opublikować wiadomość odpowiednio o treści `on` lub `off`. Stan przełącznika jest cyklicznie publikowany przez miernik pod tematem `shellies/shellyplug-s-<device_id>/relay/0`. Tutaj, podobnie jak w Shelly 3EM, gdy pod tematem `shellies/command` zostanie opublikowana wiadomość o treści `announce` to urządzenie opublikuje wiadomość w formacie JSON, której treść będzie zawierała informacje ogólne o urządzeniu.

Zamysłem w projekcie jest, aby jeden taki miernik zużycia energii przypadał na każdą koparkę osobno.

2.4. Kontrolowanie pracy koparek

W tej części projektu zostało zastosowane autorskie rozwiązanie. Koparka kryptowalut to zazwyczaj zwykły komputer z podłączonymi wieloma kartami graficznymi oraz z możliwością podłączenia standardowego okablowania komputera. Płyty główne posiadają piny konfiguracyjne, które odpowiadają m.in. za włączenie/zrestartowanie komputera przyciskiem oraz pin diody LED sygnalizującej pracę komputera.



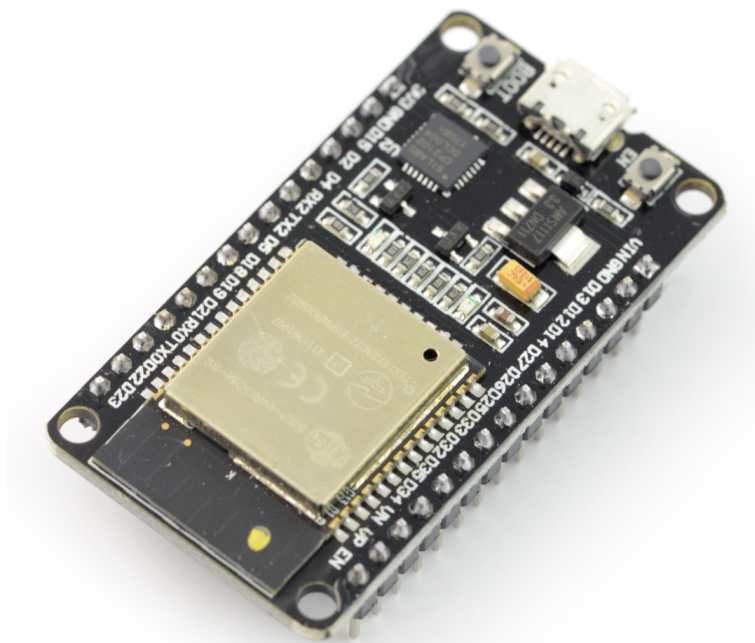
Rysunek 2.3: Przykładowa koparka kryptowalut

Autor projektu zdecydował się na zaprojektowanie własnego układu wbudowanego, którego zadaniami będą:

- włączanie/wyłączanie koparki za pomocą pinu POWER,
- restartowanie koparki za pośrednictwem sterowania pinem RESET,
- monitorowania działania koparki za pomocą pinu LED,
- wykonywanie poleceń sterowania koparek otrzymanych z głównego serwera projektu,
- zgłaszanie stanów i sytuacji niepożądanych do głównego serwera, np. nieudana próba uruchomienia, przerwanie działania koparki.

Jaka motywacja stoi za tym, aby kontrolować koparki na poziomie sprzętowym? Należy się przyjrzeć, w jaki sposób można skonfigurować komputer, aby wydobywał kryptowalutę. Najłatwiejszym sposobem jest zainstalowanie dowolnego systemu operacyjnego z rodziny Windows/Linux i uruchomienie w wierszu linii poleceń oprogramowania „wydobywającego kryptowalutę”. W tym przypadku zdalne włączanie komputera można zrealizować za pomocą wbudowanej w BIOS funkcji Wake-On-Lane, a do wyłączenia pokusić się o stworzenie prostej usługi sieciowej, która po otrzymaniu odpowiedniego polecenia zamyka system. Niestety, takie rozwiązanie ogranicza nam potencjalne wykorzystanie innych form wydobywania kryptowalut, m.in. serwisów (np. SimpleMining [9], HiveOS [10]), które na podstawie rozwiązań chmurowych, oferują zdalny podgląd i konfigurację urządzeń. Wypada też wspomnieć, że osoby wydobywające kryptowaluty często podkreślają parametry pracy swojego sprzętu, aby zmaksymalizować osiągnięty zysk. Niejednokrotnie może prowadzić to do problemu, w którym koparka z pozornie poprawnie dobranymi parametrami działania, po dłuższym czasie staje się niestabilna i się zawiesza. W takim przypadku zdalna metoda sterowania stanem urządzenia staje się nieskuteczna i aby przynajmniej wyłączyć sprzęt, wymagana jest ingerencja użytkownika koparki. Dlatego fizyczne wykorzystanie pinów na płycie głównej komputera za pomocą dedykowanego układu okazuje się najbardziej uniwersalnym i skutecznym sposobem sterowania koparką.

Koncepcyjnie w ramach tego projektu ten dedykowany układ wbudowany będzie nazywany **Guardem**. Jako jednostka wykonawcza Guarda została dobrana płytka wbudowana ESP32 [11], prócz kilkudziesięciu wyprowadzonych pinów ogólnego przeznaczenia układ posiada wbudowany moduł do komunikacji WiFi. Od strony sprzętowej w urządzeniu na każdą koparkę będzie przypadał, jeden zestaw przekaźników dwukanałowych do sterowania pinami POWER/RESET, transoptor do odczytywania stanu diody LED oraz 3 piny ogólnego przeznaczenia układu ESP32. Ostatecznie Guard został zaprojektowany tak, aby obsługiwać najwyżej 4 koparki jednocześnie.



Rysunek 2.4: Układ wbudowany ESP32

Guard również będzie korzystał z protokołu MQTT do komunikacji z głównym serwerem. Wyszczególnione tematy, pod którymi są publikowane i otrzymywane wiadomości, zostaną opisane w rozdziale prezentującym implementację rozwiązania.

2.5. Gromadzenie pomiarów danych prądowych

Aby główny serwer projektu mógł odtworzyć dane istotnie potrzebne do oszacowania, ile energii może wykorzystać do zasilania koparek, autor projektu postanowił zintegrować system z silnikiem bazodanowym. Wybraną, w tym celu, bazą danych został PostgreSQL, wersja 11.14. Na każdy miesiąc będą przypadać 3 tabele z następującymi danymi:

2.5.1. Dane odczytywane w rozdzielniczy prądowej

W tabelach o nazwie `switchboard_<month>_<year>` są zapisywane dane związane z monitorowanymi parametrami przez urządzenie Shelly 3EM. Atrybuty tabeli:

- `ts` – znacznik czasu wprowadzenia wiersza do tabeli i jednocześnie klucz główny.
- `energy_consumed_Wmin_<i>` – wartość energii pobranej z sieci energetycznej na fazie `<i>` wyrażona w wato-minutach.

- `energy_returned.Wmin_<i>` – wartość energii oddanej do sieci energetycznej na fazie `<i>` wyrażona w wato-minutach.
- `total_consumed.Wh_<i>` – całkowita wartość energii pobranej z sieci energetycznej na fazie `<i>` wyrażona w wato-godzinach.
- `total_returned.Wh_<i>` – całkowita wartość energii oddanej do sieci energetycznej na fazie `<i>` wyrażona w wato-godzinach.

2.5.2. Ogólne dane zużycia energii przez koparki

W tabelach o nazwie `miners_<month>_<year>` są gromadzone szczegółowe dane na temat ogólnego zużycia prądu przez wszystkie koparki dostępne w systemie. Atrybuty tabeli:

- `ts` – znacznik czasu wprowadzenia wiersza do tabeli i jednocześnie klucz główny.
- `name` – nazwa koparki, której dotyczą pomiary.
- `energy_consumed.Wmin` – wartość energii zużytej przez koparkę wyrażona w wato-minutach.
- `phase` – faza, na której odbył się pomiar.
- `power_W` – chwilowa moc koparki wyrażona w watach.

2.5.3. Zużycie energii uzyskanej z sieci energetycznej przez koparki

W tabelach o nazwie `miners_grid_<month>_<year>` też są zapisywane dane odnoszące się do pracy koparek, ale dotyczą one wyłącznie tego, ile prądu pobranego z publicznej sieci energetycznej zostało przez nie wykorzystane. Atrybuty tabeli:

- `ts` – znacznik czasu wprowadzenia wiersza do tabeli i jednocześnie klucz główny.
- `energy_consumed.Wmin` – wartość energii pobranej przez koparki z sieci energetycznej wyrażona w wato-minutach.
- `phase` – faza, na której odbył się pomiar.

2.6. Centralny serwer projektu

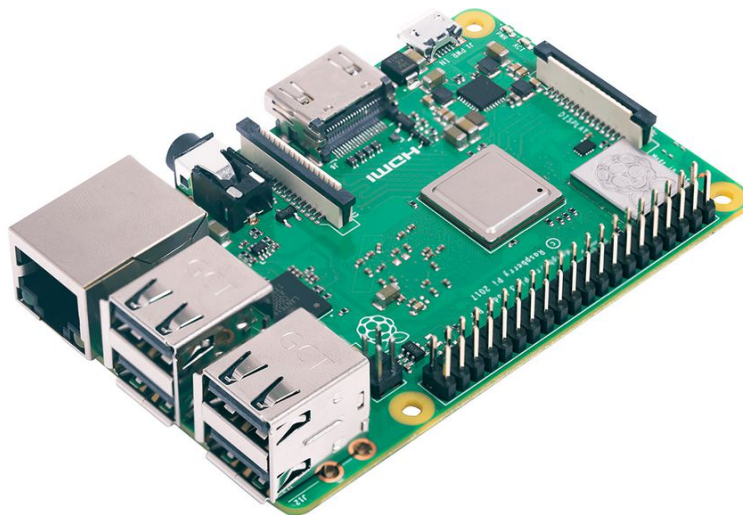
Główny serwer monitorujący wszystkie urządzenia pomiarowe i zarządzający koparkami za pomocą Guardów został nazwany koncepcyjnie **Mithra**. Autor projektu zdecydował się zaimplementować Mithre w zyskującym z biegiem czasu na popularności języku Rust, wersja 1.58. Główne zadania serwera:

- wczytywanie konfiguracji systemu: serwer MQTT, serwer bazy danych, lista urządzeń pomiarowych, lista Guardów, warunki rozliczania prosumenta.
- przetwarzanie danych pomiarowych i zapisywanie ich w bazie danych.
- komunikowanie się z Guardami i nadzorowanie pracy koparek.
- planowanie w czasie rzeczywistym zestawu koparek, które mają pracować, na podstawie odczytanych parametrów prądowych.
- umożliwienie użytkownikowi ingerowania w działanie systemu.

Szczegóły realizacji poszczególnych zadań zostaną dokładnie omówione w rozdziale opisującym implementację Mithry.

2.7. Tunelowanie VPN

Aby dać autorowi pracy możliwość rozwijania projektu niezależnie od miejsca, w jakim przebywał, został wykorzystany serwer VPN skonfigurowany w chmurze, z którym komunikował się lokalny komputer, pozwalający na zdalny dostęp do wszystkich urządzeń wykorzystanych w systemie. Do utworzenia tunelu VPN użyto oprogramowania WireGuard [13]. Urządzenie, które odgrywało rolę lokalnego routera ze skonfigurowanym klientem VPN w lokalnej sieci, to Raspberry Pi 3B+ [14] z zainstalowanym systemem Raspberry Pi OS [15] opartym na jądrze Linux.



Rysunek 2.5: Minikomputer Raspberry Pi 3B+

Rozdział 3.

Implementacja rozwiązań

W tym rozdziale zostanie omówione, w jaki sposób zostały zaimplementowane poszczególne wymagania systemu. Kod źródłowy projektu został zintegrowany z popularnym systemem kontroli wersji Git oraz platforma zdalną Github. Kod źródłowy projektu jest dostępny pod adresem:

<https://github.com/receek/Engineering-degree-diploma>

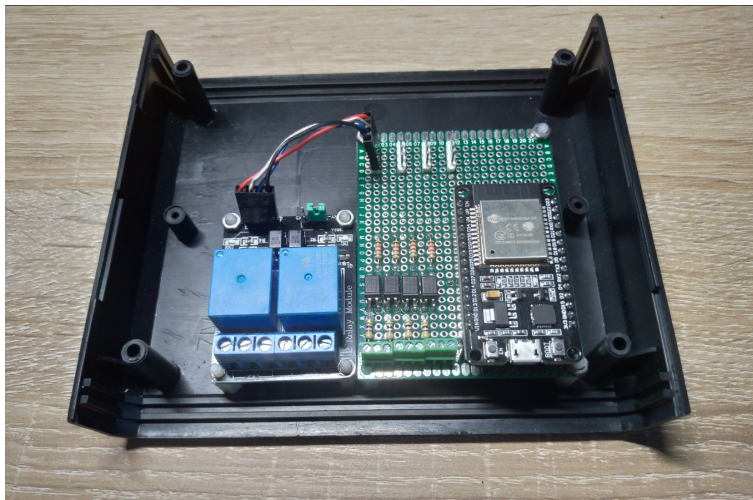
Część programowa została w całości zaimplementowana w edytorze kodu źródłowego Visual Studio Code [16], z wykorzystaniem kolejnych rozszerzeń edytora usprawniających pracę nad projektem:

- Arduino [17] – dodatek zintegrowany z platformą Arduino IDE, umożliwia z poziomu edytora kodu kompilowanie i wczytanie do pamięci płytki wbudowanej ESP32 implementacji programowej.
- C/C++ IntelliSense [18] – wsparcie dla edycji kodu C/C++.
- Rust-Analyzer [19] – wsparcie dla edycji kodu Rust.

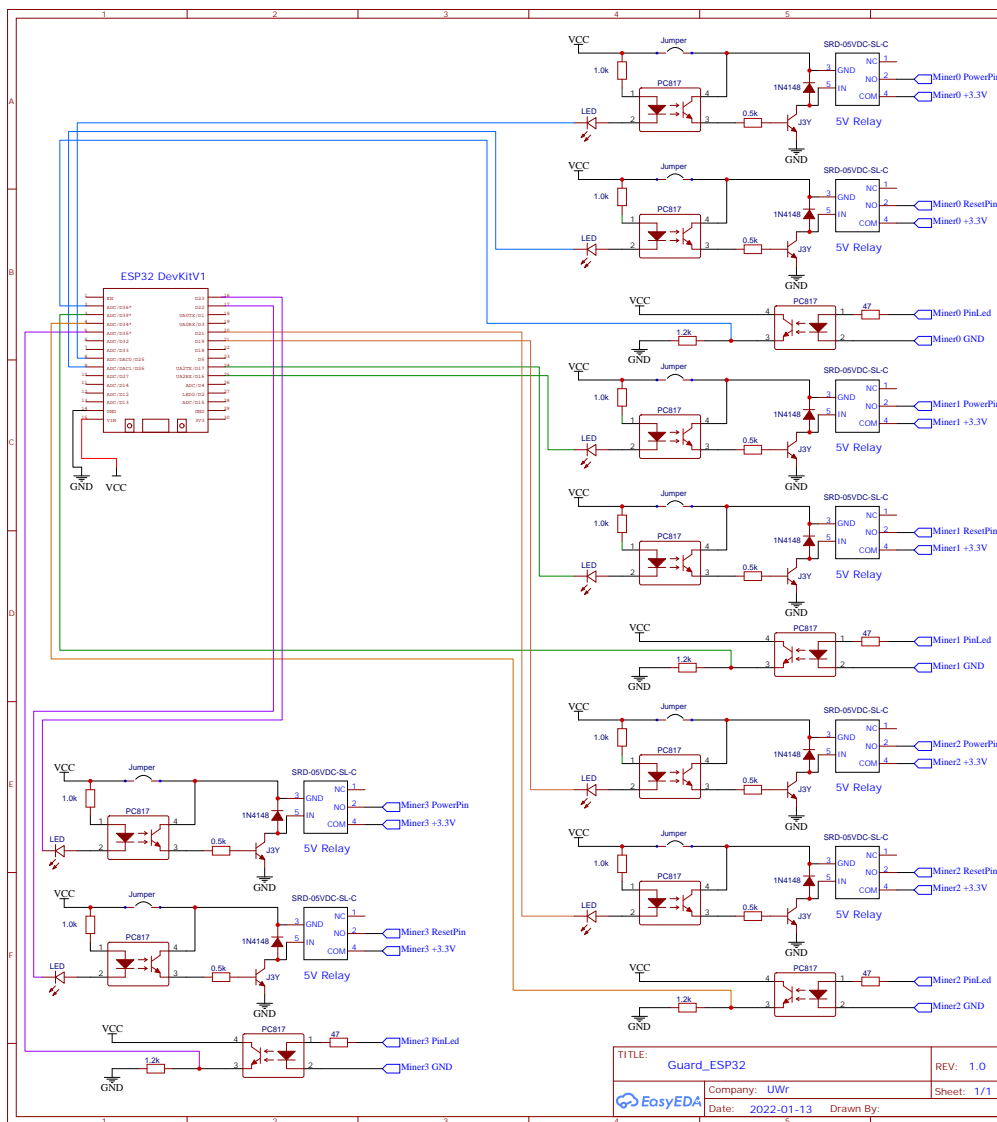
3.1. Implementacja - Guard

3.1.1. Część sprzętowa

Prototyp Guard'a został skonstruowany i utrwalony na uniwersalnej płytce montażowej przeznaczonej do elementów elektronicznych. Urządzenie zostało zaprojektowane tak, aby móc obsłużyć do 4 koparek. Na kolejnej stronie przedstawiono zdjęcie podglądowe prototypu oraz jego schemat elektryczny.



Rysunek 3.1: Prototyp Guard'a



Rysunek 3.2: Schemat Guard'a ESP32

3.1.2. Część programowa

Omówione teraz zostaną ważniejsze części implementacji programu wgranego na płytkę ESP32. Program układu został zaimplementowany w językach C/C++ i bazuje na platformie programistycznej Arduino IDE, wersja 1.8.16 [20], oferującej zestaw bibliotek do obsługi protokołu MQTT [21].

Hierarchia plików

Implementacja jest dostępna pod ścieżką `/dev/guards/ESP32`

```
/dev/guards/ESP32
├── main.ino
├── src
│   ├── globals.hpp
│   ├── miner.cpp
│   ├── miner.hpp
│   ├── timer_wrapper.cpp
│   ├── timer_wrapper.hpp
│   └── utils.hpp
```

Konfiguracja ogólna

W pliku `globals.hpp` znajduje się konfiguracja ogólna urządzenia, czyli parametrów potrzebnych do nawiązania komunikacji sieciowej WiFi, konfiguracja serwera MQTT, zestaw pinów przypadających na każdą koparkę, czasy sterowania stycznikami, konfiguracja timera sprzętowego oraz część tematów MQTT. Przed uruchomieniem Guarda użytkownik systemu musi uzupełnić poprawnymi danymi konfiguracji WiFi, brokera MQTT oraz nazwę Guarda i załadować program do urządzenia, aby umożliwić poprawną komunikację z głównym serwerem projektu.

```
const char WIFI_SSID[] = "subnet_SSID";
const char WIFI_PASSWORD[] = "secret_WiFi";

const IPAddress IP_MQTT_BROKER(127, 0, 0, 1);
const int PORT_MQTT_BROKER = 1883;
const char USER[] = "broker";
const char PASSWORD[] = "secret_broker";

const char DEV_ID[] = "Guard00";
```

Obsługa timera sprzętowego

W plikach `timer_wrapper.hpp/cpp` znajduje się implementacja klasy `TimerWrapper` pozwalającej na odczytywanie wartości timera sprzętowego i operowaniu na tych wartościach, umożliwiając pomiar upływu czasu dla różnych zadań programu. Numer wykorzystanego timera sprzętowego i preskaler zostały zdefiniowane w pliku `globals.hpp`. Preskaler został dobrany tak, aby licznik timera był inkrementowany co 1 mikrosekunde. Klasa udostępnia 2 metody:

- `getTimestamp()` – zwraca aktualną wartość licznika.
- `isTimeElapsed(ts, time_ms)` – zwraca wartość logiczną, oznaczającą czy od czasu `ts` minęło `time_ms` milisekund.

Reprezentacja koparki

W plikach `miner.hpp/cpp` znajduje się implementacja klasy pozwalającej na obsługę zadań dotyczących koparek. Klasa przechowuje wartości związane z daną koparką (nazwa, zestaw pinów, aktualny stan koparki) oraz implementuje metody pozwalające na wykonywanie poleceń sterowania koparką i monitorowania jej stanu.

Pętla startowa

Zanim `Guard` rozpocznie wykonywanie pętli głównej programu, zostaje wykonana procedura startowa, w której skład wchodzi następujące czynności:

- Skonfigurowanie komunikacji WiFi.
- Nawiązanie komunikacji z brokerem MQTT.
- Opublikowanie wiadomości MQTT do głównego serwera, z żądaniem przesłania konfiguracji urządzenia.
- Oczekiwanie na wiadomość z konfiguracją.
- Konfiguracja koparek i subskrybowanie wszystkich wymaganych tematów.

Pętla główna programu

W pętli głównej programu cyklicznie są wykonywane następujące czynności:

- Uruchomienie pętli postępu klienta MQTT, czyli sprawdzenie, czy klient nie odebrał wiadomości.
- Sprawdzenie dwóch flag:

- `runRestart` – jeśli flaga jest ustawiona to Guard musi się uruchomić ponownie
- `runAnnounce` – jeśli flaga jest ustawiona to Guard musi opublikować wiadomość z konfiguracją.
- Sprawdzenie, czy nastąpiło żądanie opublikowania stanu jakiejś koparki, jeśli tak, następuje realizacja żądania.
- Sprawdzenie, czy nastąpiło wydane przez serwer główny polecenie wobec jakiejś koparki.
- Nadzorowanie postępu wykonywania poleceń, jeśli jakaś koparka jest w trakcie wykonywania.
- Nadzorowanie stanu koparek, które nie wykonują żadnych poleceń, czy ich stan jest adekwatny do stanu pinu LED na płycie głównej komputera.
- Wysłanie wiadomości `ping` po upływie określonego czasu.

3.1.3. Tematy MQTT obsługiwane przez Guard'a

W tematach `<guard_id>` będzie oznaczać nazwę Guarda, a `<miner_id>` nazwę koparki, której konkretnie dotyczy wiadomość.

Temat: `guards/started`

Pod tym tematem Guard sygnalizuje głównemu serwerowi, że jest w fazie inicjalizacji i oczekuje na przesłanie wiadomości z konfiguracją koparek.

Temat: `guards/announce`

Guard publikuje wiadomość w formacie JSON, w której zawiera:

- swoją nazwę,
- rodzaj urządzenia (ESP32),
- wartość tablicową z danymi o koparkach, jakie w obecnej chwili obsługuje, w skład jednej pozycji wchodzi:
 - nazwa koparki,
 - zestaw pinów, jaki przypada na koparkę (wartość 0-3).

Tematy: `guards/command`, `guards/<guard_id>/command`

Pod tymi tematami Guard obsługuje 2 rodzaje wiadomości:

- `announce` – polecenie wysłania wiadomości w formacie JSON opisującej konfigurację Guarda,
- `reset` – komenda zrestartowania Guarda.

Temat: `guards/<guard_id>/config`

Pod tym tematem Guard oczekuje na wiadomość z konfiguracją wysłaną przez główny serwer. Format wiadomości musi być następujący:

```
"<k> <miner0_id> <miner0_pinset> <miner1_id> <miner1_pinset>..."
```

gdzie `<k>` to liczba koparek, które będzie obsługiwał Guard, a pary wartości `<miner_id>` `<miner_pinset>` oznaczają kolejno, nazwę koparki i nr. zestawu pinów, do jakich jest podłączona.

Temat: `guards/<guard_id>/configured`

W tym temacie Guard oświadcza głównemu serwerowi, że zakończył proces inicjalizacji i wykonuje główną pętlę programu.

Temat: `guards/<guard_id>/ping`

Pod tym tematem Guard cyklicznie sygnalizuje głównemu serwerowi, że jest osiągalny w sieci lokalnej.

Temat: `guards/<guard_id>/miners/<miner_id>`

Pod tym tematem serwer główny może zlecać polecenia do wykonania wobec koparki `<miner_id>`. W zawartości wiadomości mogą znajdować się następujące polecenia:

- `PowerOn` – uruchamia koparkę za pośrednictwem pinu POWER, symuluje zwykłe wciśnięcie przycisku zasilania.
- `PowerOff` – wyłącza koparkę za pośrednictwem pinu POWER, symuluje wciśnięcie przycisku zasilania, jednocześnie skutkując poprawnym zamknięciem systemu i odcięciem zasilania.
- `HardStop` – wyłącza koparkę za pośrednictwem pinu POWER w sposób wymuszony, zwieranie pinu przez ponad 5 sekund powoduje natychmiastowe odcięcie zasilania od urządzenia i niekontrolowane zaprzestanie działania systemu.

- **Reset** – powoduje restart koparki poprzez pin RESET.
- **HardReset** – również powoduje restart koparki, ale w sposób wymuszony, jest tutaj symulowanie wykonania komend kolejno **HardStop** i **PowerOn**.
- **StateReport** – polecenie skutkuje wysłaniem wiadomości do głównego serwera z informacją, w jakim stanie jest koparka.

Temat: guards/<guard_id>/miners/<miner_id>/command

Pod tym tematem Guard raportuje do głównego serwera o sukcesie lub niepowodzeniu wcześniej zleconego polecenia. Wiadomość przyjmuje postać "command=<command_status>, state=<miner_state>", gdzie <command_status> oznacza, jaki jest status wykonania polecenia, to pole może przyjąć wartości:

- **DONE** – raportuje powodzenie wykonania polecenia.
- **FAILED** – raportuje niepowodzenie wykonania polecenia.
- **BUSY** – sygnalizuje, że główny serwer próbował zlecić polecenie w trakcie wykonywania innego polecenia.
- **DISALLOWED** – oznacza, że zlecone polecenie jest nieadekwatne do stanu w jakim jest obecnie koparka (np. próba wyłączenia niepracującej koparki).
- **UNDEFINED** – oznacza, że serwer wysłał niezdefiniowane polecenie.

Natomiast <miner_state> oznacza, w jakim stanie obecnie jest koparka. Wszystkie możliwe wartości tego pola zostaną opisane w sekcji omawiającej następny temat.

Temat: guards/<guard_id>/miners/<miner_id>/state

Pod tym tematem następuje zaraportowanie stanu koparki, które było wcześniej zażądane poleceniem **StateReport**. Stany, jakie może przyjąć koparka i ich znaczenie:

- **PoweredOff** – koparka jest wyłączona.
- **Starting** – postępuje proces włączania koparki.
- **Running** – koparka jest włączona.
- **Stopping** – postępuje proces wyłączania koparki.
- **HardStopping** – postępuje wyłączenie koparki w sposób wymuszony.

- `Restarting` – postępuje resetowanie koparki.
- `HardRestarting` – postępuje resetowanie koparki w sposób wymuszony.
- `Aborted` – koparka działała poprawnie lub była uruchamiana, ale jej działanie zostało przerwane.
- `Unreachable` – koparka powinna być wyłączona, z jakiegoś powodu jednak jest wciąż uruchomiona i nie reaguje na polecenia wyłączenia.

Temat: `guards/<guard_id>/miners/<miner_id>/alert`

Pod tym tematem zgłaszane są niepożądane wydarzenia związane z działaniem koparek. Treść wiadomości może przyjąć następującą postać:

- `PoweredOff` – koparka była włączona, lecz jej działanie zostało nieoczekiwanie przerwane.
- `PoweredOn` – koparka była wyłączona, lecz nieoczekiwanie została uruchomiona.

3.2. Implementacja - główny serwer systemu Mithra

W tej sekcji zostanie omówiony główny serwer całego systemu, który będzie odpowiedzialny wykorzystanie nadmiarowej energii do zasilania koparek. Program został skompilowany z pomocą narzędzi wchodzących w skład pakietu Cargo [23], dedykowanego do platformy języka Rust, który m.in. jest managerem bibliotek języka dostępnych na platformie `crates.io`.

Hierearchia plików

Kod źródłowy jest dostępny pod ścieżką `dev/mithra`

```
dev/mithra
├── Cargo.toml
├── src
│   ├── main.rs
│   └── system
│       ├── database
│       │   ├── mod.rs
│       │   └── queries.rs
│       ├── handlers.rs
│       ├── mod.rs
│       └── structs.rs
```

Plik `Cargo.toml` służy do zdefiniowania bibliotek potrzebnych w systemie, które są pobierane przez Cargo ze zdalnego repozytorium i kompilowane lokalnie.

3.2.1. Uruchomienie programu

Program jest uruchamiany z dwoma wymaganymi parametrami:

- `-g, --general` – ścieżka do pliku w formacie `ini` z parametrami serwerów i dane opisujące okres rozliczeniowy.
- `-d, --devices` – ścieżka do pliku w formacie `yaml` z danymi opisującymi konfigurację Guard'ów i urządzeń pomiarowych.

3.2.2. Konfiguracja systemu

Zostaną tutaj omówione pliki konfiguracyjne aplikowane do poprzednio opisanych parametrów programu.

Plik konfiguracyjny aplikowany do parametru `-g`

W tym pliku zostają zawarte wszystkie potrzebne dane do nawiązania komunikacji z serwerem bazodanowym i brokrem MQTT. Dodatkowo są tutaj wprowadzane dane opisujące warunki rozliczeniaa prosumenta przez dostawcę energii. Przykład konfiguracji:

```
[Database]
Host = 127.0.0.1
Port = 5432
DatabaseName = mithra
User = mithra
Password = secret_psq1

[Mqtt_server]
Host = 127.0.0.1
Port = 1883
User = broker
Password = secret_broker

[Contract]
YearStart = 2022
MonthStart = 1
BillingPeriod = 12 #months
RecoveryRatio = 0.8
```

Parametr `RecoveryRatio` oznacza współczynnik oddanej energii, która można wykorzystać za darmo przed końcem okresu rozliczeniowego.

Plik konfiguracyjny aplikowany do parametru -d

W tym pliku zawarte zostają dane opisujące wszystkie urządzenia Shelly, Guardy i koparki wchodzące w skład systemu. Poprawnie skonfigurowany plik musi zawierać:

- nazwę identyfikującą urządzenie Shelly 3EM,
- listę Guardów.

W ramach opisu konfiguracji Guarda wchodzi dane:

- nazwa Guarda,
- typ urządzenia,
- lista koparek, które będzie obsługiwał.

Zestaw danych opisujących koparkę musi zawierać:

- nazwę koparki,
- zestaw pinów Guarda, do których jest podłączona,
- nazwa urządzenia pomiarowego,
- numer fazy, z której czerpie zasilanie,
- szacunkowa moc koparki w watach.

Wszystkie nazwy urządzeń Shelly, Guardów i koparek muszą być unikalne w obrębie całego systemu. Przykład konfiguracji:

```
switchboard:
  id: shellyem3-id
guards:
  - id: Guard00
    type: ESP32
  miners:
    - id: Miner00
      pinset: 0
      plug: shellyplug-s-0
      phase: 0
      consumption: 200
    - id: Miner01
      pinset: 1
      plug: shellyplug-s-1
      phase: 1
      consumption: 400
```

3.2.3. Etap wstępny działania programu

Program po uruchomieniu wczytuje konfigurację i sprawdza jej poprawność. Jeśli wszystkie parametry zostały podane poprawnie, to następuje wykonanie kolejnych procedur:

- Potwierdzenie osiągalności serwerów – program sprawdza, czy serwer bazodanowy oraz broker MQTT są osiągalne w sieci.
- Ustalenie osiągalności urządzeń – następuje opublikowanie wiadomości pod tematami `shellies/command`, `guards/command` o treści `announce`, w celu ustalenia, jakie urządzenia Shelly i Guardy aktualnie działają. Kolejno program oczekuje na wiadomości od urządzeń.
- System wyznacza pracujące Guardy, z nieaktualną konfiguracją względem tej podanej w plikach konfiguracyjnych Mithry i je resetuje.
- Sprawdzenie schematu bazy danych – program sprawdza, czy w bazie danych znajdują się wszystkie potrzebne tabele, w razie potrzeby są one tworzone.
- Pobranie informacji o pomiarach prądowych z bazy danych – program komunikuje się z bazą danych, aby uzyskać następujące informacje:
 - Stan urządzenia Shelly 3EM na początku okresu rozliczeniowego,

- Zużycie prądu przez koparki od początku okresu rozliczeniowego aż do obecnego czasu systemu.

Po tym etapie następuje proces uruchomienia wątków pomocniczych.

3.2.4. Podział zadań pomiędzy wątki

Autor w procesie projektowym programu zdecydował rozdzielić zadania głównego serwera systemu pomiędzy kilka wątków. Zanim jednak nastąpi przedstawienie tego podziału, należy omówić kilka ważnych funkcjonalności związanych z komunikacją międzywątkową i klientem MQTT wykorzystanych w implementacji Mithry.

Komunikacja międzywątkowa

Do zrealizowania komunikacji pomiędzy wątkami została wykorzystana struktura danych wbudowana w bibliotekę standardową języka Rust – `channel` [24]. Jest to kolejka FIFO typu multi-producer, single-consumer. Konstruktor struktury danych zwraca dwa obiekty – nadawcę i odbiorcę. Obiekt nadawcy może być kopiowany i rozdysponowany pomiędzy wiele wątków, podczas gdy instancja obiektu odbiorcy może być tylko jedna. Nadawanie i odbieranie danych ze struktury może zakończyć się niepowodzeniem, gdy któryś z końców kanału komunikacji zostanie zamknięty, czyli odpowiednio wszystkie obiekty nadawcze lub jedyny obiekt odbiorcy zostaną usunięte. Sytuacja niepowodzenia nadania lub odbioru danych musi zostać właściwie obsłużona przez program. Implementacja obiektu odbiorcy oferuje kilka metod odbioru danych:

- Oczekiwanie z blokowaniem wątku do momentu pojawienia się danych w kanale komunikacyjnym.
- Oczekiwanie na dane z blokowaniem wątku określoną ilość czasu lub do określonego punktu w czasie. Jeśli w kanale komunikacyjnym nie pojawiły się dane, a czas oczekiwania upłynął, to wątek wznowia działanie. Fakt braku odebrania danych musi zostać właściwie obsłużony przez program.

Klient MQTT

W programie została wykorzystana jedna z implementacji klienta MQTT [25] z pośród dostępnych na platformie `crates.io`. Po ustawieniu wszystkich parametrów brokera, obsługa komunikacji protokołu MQTT skupia się wokół dwóch obiektów:

- Obiekt klienta – pozwala na zarządzanie obiektem połączenia, czyli za jego pośrednictwem ustawia się subskrybowane tematy oraz zamyka połączenie z

brokerem MQTT. Za pomocą tego obiektu można również publikować wiadomości.

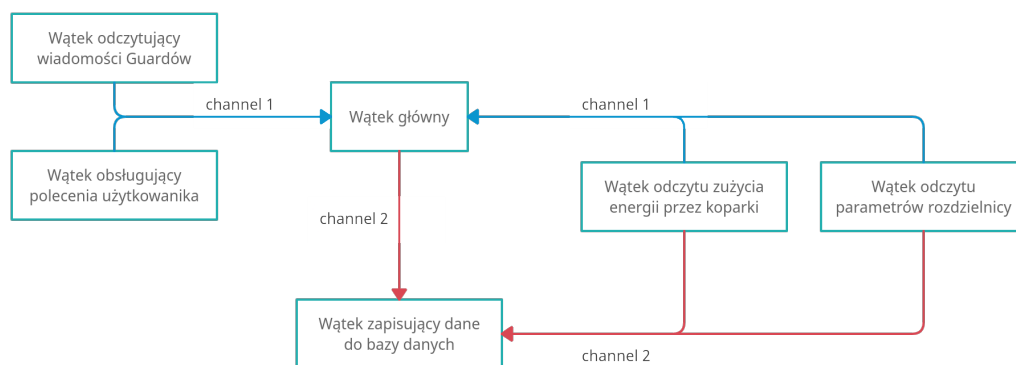
- Obiekt połączenia – z jego pomocą odbiera się wszystkie pakiety wysyłane pomiędzy klientem i brokerem MQTT, w tym wiadomości publikowane na subskrybowane tematy przez obiekt klienta. Odczyt pakietów jest realizowany z poziomu obiektu połączenia w sposób blokujący wątek do momentu otrzymania danych.

Podział zadań

Poszczególne zadania Mithry zostały podzielone według następującego schematu pomiędzy wątki:

- Odczyt danych prądowych z rozdzielnic – wątek ma za zadanie odczytywać dane zużycia/produkcji prądu w gospodarstwie domowym publikowane przez urządzenie Shelly 3EM. Zgromadzone dane zostają kolejno przekazane do wątku głównego oraz do wątku zapisującego informacje w bazie danych.
- Odczyt zużycia prądu przez koparki – wątek odczytuje wszystkie wiadomości publikowane przez urządzenia Shelly PlugS, czyli będzie monitorował dane zużycia prądu przez koparki. Jak w poprzednim wątku zgromadzone dane też zostają przekazane do wątku głównego oraz do wątku zapisującego informacje w bazie danych.
- Zapisywanie informacji do bazy danych – zadaniem wątku jest utrzymywanie połączenia z bazą danych systemu i zapisywanie informacji prądowych.
- Odbieranie wiadomości Guardów – wątek odpowiada za odbieranie wszystkich wiadomości wysyłanych przez Guardy, dane wstępnie formatuje i przekazuje do wątku głównego.
- Odbieranie poleceń użytkownika – odpowiada za odbieranie poleceń użytkownika systemu i przekazywanie ich do wątku głównego
- Wątek główny – odbiera informacje od innych wątków, przetwarza je i część nich zapisuje do bazy danych, zarządza połączeniami klientów MQTT, śledzi parametry prądowe, na podstawie których podejmuje decyzje o włączaniu/wyłączaniu koparek, wydaje polecenia Guardom i nadzoruje ich działanie, reaguje na wydarzenia niepożądane.

Przepływ komunikacji między wątkami został zaprezentowany na poniższym obrazku. Program korzysta z dwóch obiektów `channel`.



Rysunek 3.3: Przepływ danych między wątkami

Na każdy wątek przetwarzający publikowane wiadomości MQTT przypada jeden obiekt połączenia. Wszystkie obiekty klienta są umieszczone w głównym wątku, który decyduje, jakie tematy mają być subskrybowane i publikuje wiadomości do urządzeń.

3.2.5. Działanie wątków odczytujących wiadomości MQTT

Wszystkie wątki odczytujące wiadomości MQTT działają na podobnej zasadzie. Wątek blokuje się na obiekcie połączenia i czeka na otrzymanie wiadomości MQTT, następnie odpowiednio ją przetwarza, formatuje i przesyła do wątku głównego (ewentualnie jeszcze do wątku obsługującego bazę danych), aby ten z kolei mógł uaktualnić stan systemu.

3.2.6. Wątek obsługujący komunikację z bazą danych

Ten wątek utrzymuje otwarte połączenie z serwerem PostgreSQL i oczekuje na dane, z obiektu `channel` w sposób blokujący, które ma umieścić w bazie danych.

3.2.7. Zadania głównego wątku

Wątek główny oczekuje na dane na obiekcie odbiorcy struktury `channel` w sposób blokujący, ale z limitem czasowym na oczekiwanie. Czas oczekiwania dobrany jest tak, aby raz na 30 sekund zostało wykonane sprawdzenie stanu wszystkich urządzeń w systemie oraz, czy polecenia zlecone Guardom przebiegają poprawnie. Natomiast raz na 5 minut następuje rozdysponowanie zasobów prądowych między koparki. Taka decyzja projektowa podyktowana jest faktem, że między fizycznym włączeniem koparki a pełnym jej uruchomieniem, czyli generowaniem cyfrowego przychodu, może upłynąć od kilkudziesięciu sekund do nawet około 2 minut. Przy

krótszym czasie między kolejnymi planowaniami zużycia zasobów energetycznych w przypadku skrajnie zmiennych warunków pogodowych, podział nadmiarowej energii mógłby skutkować jedynie jałowym włączeniem koparki, tylko po to, aby w następnym cyklu ją wyłączyć. W ciągu 5 minut wątek główny gromadzi dane produkcji/zużycia energii w gospodarstwie domowym i w momencie rozdysponowania nadmiarowych zasobów zakłada, że warunki pogodowe nie zmieniają się w stopniu znacznym przez kolejne 5 minut. Sprawdzenie stanu wszystkich urządzeń w systemie polega na kilku rzeczach:

- Sprawdzenie stanu Guardów i przyporządkowanych im koparek. Jeśli urządzenia wykonują wobec jakiejś koparki polecenie uruchomienia/zatrzymania to nadzorowane jest czy przebiega ono poprawnie. Guard musi zaraportować do głównego serwera, czy wysłana komenda zakończyła się wykonywać z zamierzonym skutkiem. W przypadku, gdy koparka zachowuje się w sposób niezamierzony, to Mithra może zlecić fizyczne odcięcie zasilania koparce, poprzez urządzenie Shelly PlugS, które zostało do niej przyporządkowane.
- Punkt pomiarowy w rozdzielniczy jest najbardziej kluczowym elementem całego systemu. Komplet wszystkich mierzonych parametrów publikuje średnio raz na minutę. Gdy Mithra stwierdzi, że urządzenie przestaje być osiągalne w sieci lokalnej, następuje wyłączenie wszystkich uruchomionych koparek.
- Mierniki zużycia energii przez koparki również publikują swoje pomiary średnio raz na minutę. Na tej podstawie Mithra wnioskuje, czy te urządzenia też są osiągalne.

Koparka może zostać wykluczona z rozdysponowania zasobów i dzieje się to w dwóch przypadkach:

- Z przyczyn występowania sytuacji niepożądanych w systemie – koparka jest niestabilna, Guard lub urządzenie Shelly PlugS jest nieosiągalne w sieci lokalnej.
- Użytkownik systemu zdecydował ją wykluczyć.

3.2.8. Wykluczanie koparek

Aby użytkownik systemu posiadał możliwość serwisowania koparek bez przerwania działania głównego serwera systemu, został zaimplementowany mechanizm wykluczania. Gdy jakaś koparka zostanie wykluczona Mithra przestaje subskrybować tematy związane z daną koparką i nie bierze jej pod uwagę przy następnym planowaniu wykorzystania zasobów. Ta możliwość również jest zrealizowana w systemie za pomocą protokołu MQTT. Oczywiście wykluczoną koparkę można z powrotem włączyć do systemu. Mithra subskrybuje temat `user/<miner_id>` pod którym można opublikować wiadomości:

- **Exclude** – następuje wykluczenie koparki `<miner_id>`.
- **Include** – koparka `<miner_id>` zostaje z powrotem włączona w działanie systemu.

3.2.9. Algorytm planowania zużycia zasobów energii

Po ustaleniu, jakie koparki są dostępne w systemie, dzieli się je na 2 grupy: koparki uruchomione i wyłączone. W każdej z tych grup rozróżnia się jeszcze, pod jakimi fazami koparki są zasilane. Algorytm planowania podzielony jest na 3 scenariusze:

1. Gospodarstwo zużyło więcej energii, niż może pobrać za darmo z sieci energetycznej.
2. Fotowoltaika wyprodukowała znacznie więcej energii, niż gospodarstwo będzie mogło zużyć do końca okresu rozliczeniowego.
3. Żaden z poprzednich scenariuszy nie nastąpił, ale fotowoltaika w obecnej chwili produkuje energię, więc możemy spróbować zużywać część energii na bieżąco.

Scenariusz 1. jest prosty – wszystkie koparki mają zostać wyłączone. Scenariusze 2. i 3. będą korzystały przy ustalaniu jakie koparki mają pracować z metody polegającej na programowaniu dynamicznym. Najpierw zostanie omówiony scenariusz 3. ponieważ scenariusz 2. wykorzystuje ten sam algorytm z małą modyfikacją.

Algorytm w scenariuszu 3.

Dynamiczny algorytm, który będzie planował zużycie energii, polega na rozwiązaniu dyskretnego problemu plecakowego. Przypomnijmy, że w tym problemie rozważamy zbiór n przedmiotów d_i , każdy o wadze w_i i wartości c_i oraz plecak o maksymalnej nośności W . Chcemy tak dobrać podzbiór przedmiotów d_i , aby ich sumaryczna wartość była jednocześnie jak największa i w sumie nie przekraczała wagi W . Problem plecakowy jest rozpatrywany w przypadku koparek w następujący sposób:

- Waga przedmiotu to moc znamionowa koparki.
- Maksymalna dopuszczalna waga W to maksymalna sumaryczna moc koparek, które mogą zostać uruchomione przy aktualnych parametrach produkcji prądu.
- Ponieważ dążymy do zmaksymalizowania zużycia nadmiarowej energii to wartość przedmiotu-koparki jest równoważna jej wadze-mocy.

Mithra monitoruje parametry prądowe z rozróżnieniem na fazy, zatem wylicza nadmiarową dostępną moc osobno na każdą fazę i rozwiązuje problem plecakowy w obrębie każdej fazy. Scenariusz 3. jest podzielony na 2 warianty:

- W przydadku, gdy względem ostatniego rozdysponowania zasobów produkcja prądu się zwiększyła, to algorytm decyduje, że włączone koparki mają pozostać włączone, a niewykorzystaną moc rozdysponuje pomiędzy wyłączone koparki.
- W sytuacji przeciwnej następuje ograniczenie liczby pracujących koparek, czyli spośród nich wybranie podzbioru, który maksymalnie zużywa mniejszą dostępną moc.

Pseudokod rozwiązania problemu plecakowego.

```
# koparka o indeksie 'i' posiada moc 'miners_power[i]'
# max_power - maksymalna dostepna moc do wykorzystania

def knapsack(miners_power[n], max_power):
    dp[max_power + 1]; # domyślnie wypełniona wartościami null
    dp[0] = -1;
    for i in [0, ..., n-1]:
        power = miners_power[i];
        for j in [max_power, ..., power]:
            if (dp[j] == null && dp[j - power] != null):
                dp[j] = i;

    # rekonstrukcja zbioru indeksów uruchomionych koparek
    id = max_power;
    running_miners = [];
    while (id > 0 && dp[id] == null):
        id--;
    while (id > 0)
        running_miners.push(dp[id]);
        id -= miners_power[dp[id]];

    return running_miners;
```

Algorytm w scenariuszu 2.

Scenariusz 3. rozpatrywał zużycie prądu w ramach każdej fazy osobno. W scenariuszu 2. istnieje dodatkowo możliwość zużywania prądu z sieci energetycznej na dowolnej fazie, zatem należy dobrać optymalny zestaw pracujących koparek, uwzględniając wszystkie 3 fazy jednocześnie. Jak jest obliczany dostępny nadmiar energii?

Mithra na podstawie monitorowanych parametrów prądowych jest w stanie wyznaczyć, ile energii zostało pobrane z sieci energetycznej i wśród tych pomiarów wyróżnia, ile energii zużyły koparki. Pozostała pobrana energia to zapotrzebowanie gospodarstwa domowego na prąd. Mithra wylicza na tej podstawie, jaka jest średnia moc gospodarstwa od początku okresu rozliczeniowego i zakłada takie samo zapotrzebowanie na prąd do końca okresu rozliczeniowego. Wyliczone parametry porównuje z ilością zwróconej energii do sieci energetycznej i stwierdza, jaka jest dostępna nadmiarowa moc do wykorzystania zasilania koparek.

Scenariusz 2. ma na celu zmaksymalizować zużycie nadmiarowej energii przed końcem okresu rozliczeniowego, dlatego zawsze będzie dobierał optymalny możliwy podzbiór koparek niezależnie od tego, które koparki są uruchomione, a które nie. Aby przedstawić, jak są dobierane koparki z uwzględnieniem wszystkich 3 faz, niech zostaną zdefiniowane następujące zmienne:

p_i – dostępna moc na fazie i wynikająca z produkcji prądu przez fotowoltaikę.

r – moc możliwa do wykorzystania z sieci energetycznej.

max_i – maksymalna możliwa moc koparek na fazie i .

max_power – maksymalna możliwa sumaryczna moc koparek.

$$max_i = p_i + r$$

$$max_power = p_1 + p_2 + p_3 + r$$

Poprawne rozdysponowanie zasobów polega na tym, aby w obrębie każdej fazy i sumaryczna moc dobranych koparek nie przekraczała wartości max_i . Wykorzystany algorytm do planowania zużycia nadmiarowej mocy będzie podobny do tego, który został wykorzystany w scenariuszu 3. Mała modyfikacja polega na tym, że element tablicy `dp[i]` prócz tego, że pamięta, jaki jest indeks dobranej koparki, to jeszcze dodatkowo zapamiętuje trzelementową tablicę $[m_1, m_2, m_3]$, w której poszczególne elementy oznaczają, ile mocy z wcześniej dobranych koparek przypada na poszczególną fazę. Pseudokod znajduje się na następnej stronie.

```
# miner[i].power- oznacza moc koparki 'i'
# miner[i].phase - oznacza fazę pod jaką jest podłączona koparka 'i'
# phase_power[i] - moc produkowana na fazie 'i'
# max_returned - maksymalna dopuszczalna moc czerpana z sieci
energetycznej

def knapsack(miners[n], phase_power[3], max_returned):
    max_phase[] = [0,0,0];
    max_power = max_returned;
    for i in [0, 1, 2]:
        max_phase[i] = phase_power[i] + max_returned;
        max_power += phase_power[i];

    dp[max_power + 1]; # domyślnie wypełniona wartościami null
    dp[0] = (-1, [0,0,0]);

    for i in [0, ..., n-1]:
        power = miners[i].power;
        p = miners[i].phase;
        for j in [max_power, ..., power]:
            if (dp[j] == null && dp[j - power] != null):
                (id, phases) = dp[j - power];
                if (phases[p] + power <= max_phase[p]):
                    phases[p] += power;
                    dp[j] = (i, phases);

    # rekonstrukcja zbioru indeksów uruchomionych koparek
    i = max_power;
    running_miners = [];
    while (i > 0 && dp[i] == null):
        i--;
    while (i > 0)
        (idx, phases) = dp[i];
        running_miners.push(idx);
        i -= miners_power[idx].power;

    return running_miners;
```


Rozdział 4.

Uruchomienie systemu

4.1. Serwer PostgreSQL i borker MQTT

W projekcie został wykorzystany serwer bazodanowy PostgreSQL [12], wersja 11.14, oraz broker MQTT Mosquitto [2], wersja 1.6.9, zainstalowany na komputerze wyposażony w system operacyjny oparty na jądrze Linux. Użytkownik na własne potrzeby może dobrać dowolny inny system operacyjny wspierany przez implementację PostgreSQL. Dostępne są również inne implementacje brokera MQTT i użytkownik również może zdecydować, którą chce zastosować.

Przed uruchomieniem Mithry w konsoli PostgreSQL należy wykonać poniższy ciąg poleceń. Parametry nazwy użytkownika, hasła i nazwy bazy danych mogą być zmienione wedle uznania. Te same parametry należy potem wpisać do plików konfiguracyjnych Mithry.

```
CREATE DATABASE mithra;  
CREATE ROLE mithra LOGIN PASSWORD 'secret';  
REVOKE ALL ON DATABASE mithra FROM mithra;  
GRANT CONNECT ON DATABASE mithra TO mithra;
```

Podobnie w przypadku brokera MQTT, użytkownik może zdecydować o postaci nazwy użytkownika i jego hasła.

4.2. Urządzenia Shelly

Użytkownik montuje wszystkie urządzenia Shelly według instrukcji producenta. Następnie konfiguruje je, aby zapewnić połączenie WiFi i komunikację z brokerem MQTT. Wszystkie dane urządzeń musi odzwierciedlić w konfiguracji Mithry.

4.3. Wgrywanie programu Guarda

Tą część projektu rozwinięto w środowisku programistycznym Arduino IDE [20], wersja 1.8.16, dlatego zaleca się użytkownikowi korzystanie z tego środowiska w celu wgrania poprawnej konfiguracji na płytkę Guarda. Wszystkie wymagane do skonfigurowania parametry znajdują się pliku `dev/guards/ESP32/src/globals.hpp`. Po zakończonej edycji użytkownik kompiluje i wgrywa program do pamięci urządzenia. Przed jego zasileniem, podłącza koparki zgodnie ze schematem elektrycznym.

4.4. Kompilacja i uruchomienie Mithry

Aby skompilować główny system, należy pobrać kompilator języka Rust [22] i narzędzie Cargo. W celu skompilowania programu należy w głównym katalogu programu `dev/mithra/` wydać polecenie `cargo build --release`. Po ukończonej kompilacji plik binarny z programem będzie się znajdował na ścieżce `dev/mithra/target/release/mithra`

Przed uruchomieniem programu należy utworzyć pliki konfiguracyjne uzupełnione:

- danymi połączenia z serwerem PostgreSQL,
- danymi połączenia z brokerem MQTT,
- warunkami umowy z dostawcą energii,
- danymi wszystkich urządzeń w systemie,

aby następnie zaaplikować je do uruchomionego programu przez wcześniej opisane parametry konfiguracyjne. Program został zaprojektowany tak, aby wszystkie komunikaty diagnostyczne były wypisywane na standardowe wyjście i wyjście błędu. Zatem możliwe jest uruchomienie programu jako usługi w systemach operacyjnych opartym na jądrze Linux (np. wykorzystując popularny menadżer `systemd`).

Bibliografia

- [1] <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>
- [2] <https://mosquitto.org/documentation/>
- [3] https://mosquitto.org/man/mosquitto_pub-1.html
- [4] https://mosquitto.org/man/mosquitto_sub-1.html
- [5] <https://shelly.cloud/knowledge-base/devices/shelly-3em/>
- [6] <https://shelly-api-docs.shelly.cloud/gen1/#shelly-3em>
- [7] <https://shelly.cloud/knowledge-base/devices/shelly-plug-s/>
- [8] [https://shelly-api-docs.shelly.cloud/gen1/
#shelly-plug-plugs-mqtt](https://shelly-api-docs.shelly.cloud/gen1/#shelly-plug-plugs-mqtt)
- [9] <https://simplemining.net/>
- [10] <https://hiveon.com/os/>
- [11] [https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
get-started/index.html](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html)
- [12] <https://www.postgresql.org/docs/11/index.html>
- [13] <https://www.wireguard.com/>
- [14] <https://www.raspberrypi.com/documentation/>
- [15] <https://www.raspberrypi.com/software/>
- [16] <https://code.visualstudio.com/>
- [17] [https://marketplace.visualstudio.com/items?itemName=
vsciot-vscode.vscode-arduino](https://marketplace.visualstudio.com/items?itemName=vsciot-vscode.vscode-arduino)
- [18] [https://marketplace.visualstudio.com/items?itemName=ms-vscode.
cpptools](https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools)
- [19] [https://marketplace.visualstudio.com/items?itemName=matklad.
rust-analyzer](https://marketplace.visualstudio.com/items?itemName=matklad.rust-analyzer)

- [20] <https://www.arduino.cc/en/software>
- [21] <https://www.arduino.cc/reference/en/libraries/mqtt/>
- [22] <https://www.rust-lang.org/tools/install>
- [23] <https://doc.rust-lang.org/cargo/index.html>
- [24] <https://doc.rust-lang.org/std/sync/mpsc/fn.channel.html>
- [25] <https://docs.rs/rumqttc/latest/rumqttc/>