

Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka techniczna**
Specjalność: **Inżynieria systemów informatycznych**

PRACA DYPLOMOWA
INŻYNIERSKA

**Projekt systemu analizy danych
pomiarowych z przydomowej instalacji
fotowoltaicznej**

**Design of a system for analyzing
measurement data from a home
photovoltaic installation**

Mikołaj Chmielecki

Opiekun pracy
dr inż. Tomasz Kubik

Streszczenie

Ekologia stała się ważnym aspektem codziennego życia na poziomie lokalnym i globalnym. Poświęcono jej wiele uwagi między innymi w dziedzinie pozyskiwania energii. Dostrzeżono, że królujące tam spalanie paliw kopalnych ma negatywny wpływ na klimat naszej planety, zaś obiecującą alternatywą są odnawialnych źródeł energii. To z kolei wpłynęło na rozwój fotowoltaiki.

W ostatnich 5 latach wdrożono wiele instalacji fotowoltaicznych w gospodarstwach domowych. Ponieważ dofinansowanie na takie inwestycje można było pozyskać z różnych europejskich i krajowych źródeł narzucających różne warunki, sporo klientów decydowało się na kilkukrotne, rozłożone w czasie działania. W efekcie powstały przydomowe instalacje składające się z komponentów pochodzących od różnych producentów, dostarczających różnych aplikacji do ich monitorowania.

Heterogeniczność dostępnych aplikacji skutecznie utrudnia zarządzanie produkcją i konsumpcją energii. Dlatego celem pracy stało się stworzenie aplikacji internetowej, która ułatwi monitorowanie i agregowanie danych z wielu instalacji fotowoltaicznych różnych producentów. Aplikacja będzie komunikowała się z systemami fotowoltaicznymi oraz licznikiem energii elektrycznej. Będzie też zapisywać pozyskane dane w bazie danych i wyświetlać je w przyjaznej dla użytkowników formie.

Słowa kluczowe: analiza danych, fotowoltaika, aplikacja internetowa

Abstract

Nowadays, ecology has become an important aspect of our lives. Obtaining energy from fossil fuels has a very negative impact on the climate of our planet. For this reason, renewable energy sources, in particular photovoltaic panels, are gaining popularity. In the last 5 years, due to various European and national subsidies, many households acquired such installations.

Due to the fact that over the course of these 5 years, these subsidies differed in certain conditions, many clients decided to make this investment several times, which was often associated with the fact that they decided to buy installations from different manufacturers. Producers of photovoltaic installations have dedicated applications to monitor the operation of these installations, which, if they have many different systems, effectively hinders their monitoring.

The aim of this work is to create a web application that will facilitate the monitoring and aggregation of data from many photovoltaic installations from different manufacturers. The application will communicate with photovoltaic systems and an electricity meter. The application will save the obtained data in a database and display it in a user-friendly form.

Keywords: data analysis, photovoltaics, web application

Spis treści

1. Wstęp	8
1.1. Wprowadzenie	8
1.2. Cel pracy	10
1.3. Zakres pracy	10
1.4. Układ pracy	11
2. Projekt	12
2.1. Inicjalizacja projektu	12
2.2. Wymagania funkcjonalne	12
2.3. Wymagania niefunkcjonalne	16
2.4. Wymagania dziedzinowe	17
2.5. Model danych	17
2.6. Obsługiwane źródła danych	19
2.6.1. Licznik energii elektrycznej Energa	19
2.6.2. Falownik Solax	21
2.6.3. Falownik SolarEdge	22
3. Implementacja	24
3.1. Architektura	24
3.2. Aplikacja serwerowa	25
3.2.1. Struktura projektu	25
3.2.2. Najważniejsze fragmenty implementacji	26
3.2.3. Zabezpieczenia	28
3.2.4. Komunikacja ze źródłami danych	30
3.2.5. Struktura interfejsu REST	31
3.3. Aplikacja kliencka	32
3.3.1. Struktura projektu	35
3.3.2. Najważniejsze fragmenty implementacji	37
3.3.3. Instrukcja użytkownika	40
4. Testy aplikacji	45
4.1. Wykorzystane technologie	45
4.2. Testy jednostkowe	45
4.3. Testy automatyczne aplikacji internetowej	45
5. Podsumowanie	46
5.1. Plany rozwoju aplikacji	46
Literatura	47
A. Instrukcja wdrożeniowa	48
A.1. Wdrożenie aplikacji serwerowej wraz z bazą danych i serwisem Selenium Grid	48
A.2. Wdrożenie aplikacji klienckiej	48
B. Opis załączonej płyty DVD	50

Spis rysunków

1.1. Schemat instalacji elektrycznej z trzema systemami fotowoltaicznymi	9
2.1. Diagram przypadków użycia systemu SolarMonitoring	13
2.2. Model konceptualny	18
2.3. Model fizyczny	18
2.4. Licznik energetyczny Energa	19
2.5. Strona logowania do panelu administracyjnego Energa	20
2.6. Widok danych pomiarowych w panelu administracyjnym Energa	20
2.7. Zdjęcia falowników	21
3.1. Architektura systemu SolarMonitoring	25
3.2. Struktura projektu	27
3.3. Struktura interfejsu REST	32
3.4. Struktura projektu aplikacji klienckiej	36
3.5. Formularze: a) logowania W-1, b) rejestracji W-2	41
3.6. Widok "W-3 O aplikacji"	41
3.7. Widok "W-6 Lista falowników"	42
3.8. Widok "W-4 Pomiary"	43
3.9. Widok "W-7 Dodawanie/edycja falownika"	43
3.10. Formularze: a) zmiany danych użytkownika W-8, b) zmiany danych logowania do panelu administracyjnego licznika W-5	44
3.11. Komunikat potwierdzenia usunięcia	44
3.12. Komponenty: a) komunikat użytkownika, b) rozwijane menu dla falowników	44

Spis tabel

2.1.	Spis danych w odpowiedzi punktu końcowego <i>getRealtimeInfo</i> interfejsu Solax Cloud	22
2.2.	Spis danych w odpowiedzi punktu końcowego <i>overview</i> interfejsu SolarEdge Cloud	23
2.3.	Spis danych w odpowiedzi punktu końcowego <i>details</i> interfejsu SolarEdge Cloud .	23
3.1.	Opis API dostarczanego przez aplikację serwerową	33

Spis listingów

3.1. Główna klasa aplikacji serwerowej	26
3.2. Plik konfiguracyjny aplikacji <i>Spring Boot</i>	27
3.3. Klasa dostarczająca metod umożliwiających szyfrowanie	28
3.4. Metoda <code>configure</code> z klasy <code>WebSecurityConfig</code>	29
3.5. Metoda <code>saveMonthlyInvertersData</code> uruchamiana zgodnie z zadaniem harmonogramem	30
3.6. Zawartość pliku <code>index.js</code>	37
3.7. Kod źródłowy komponentu definiującego logikę związaną z wyborem odpowiedniego widoku w zależności od stanu uwierzytelnienia użytkownika	37
3.8. Kod źródłowy głównego komponentu aplikacji po zalogowaniu – App	38
3.9. Kod źródłowy komponentu <code>AlertInCorner</code>	39
3.10. Funkcja <code>loginUser</code> pobierająca z serwera token JWT	39
3.11. Fragment pliku <code>Dashboard.js</code> odpowiedzialny za renderowanie wykresu	40

Skróty

PCL (ang. *Power Line Communication*)

FR (ang. *Functional Requirement*)

UX (ang. *User Expierience*)

ORM (ang. *Object Relational Mapping*)

JWT (ang. *JSON Web Token*)

REST (ang. *Representational State Transfer*)

API (ang. *Application Programming Interface*)

DOM (ang. *Document Object Model*)

Rozdział 1

Wstęp

1.1. Wprowadzenie

Przez tysiące lat rozwoju ludzkiej cywilizacji Ziemię i jej naturalne zasoby niemal bez ograniczeń eksplorowano i wykorzystywano celem poprawy jakości życia oraz spełniania potrzeb jej mieszkańców. Obecnie, jak nigdy wcześniej, wyraźnie zarysowała się granica, przekroczenie której może zagrozić ludzkiej egzystencji. Mowa tu o zjawiskach towarzyszących nie tylko produkcji żywności i dóbr materialnych, ale również wytwarzaniu energii oraz ich negatywnym wpływie na środowisko. Wystarczy pochylić się nad efektem cieplarnianym, zanieczyszczeniem powietrza oraz wody, degradacją gleb i innych zagrożeniach, by zdać sobie sprawę z powagi sytuacji. Paradoksalnie za taki stan rzeczy odpowiada sam człowiek. I człowiek może tę sytuację zmienić, jeśli tylko zacznie myśleć o otaczającym go środowisku jak o dobru wspólnym, przekazywanym z pokolenia na pokolenie.

Szczególną dziedziną, osiągającą ostatnio pewne sukcesy na polu ekologii, jest wytwarzanie energii. Choć aktualnie do wytwarzania prądu i ciepła wykorzystuje się w głównej mierze paliwa kopalne, co skutkuje ponad 40%-wym ich udziałem w emisji dwutlenku węgla na świecie [4], to jednak powoli zastępowane są one zeroemisyjnymi źródłami odnawialnymi. Znaczące postępy osiągnęła tu energetyka słoneczna, w tym fotowoltaika.

Fotowoltaika jest dziedziną nauki, która zajmuje się konwersją promieniowania słonecznego na energię elektryczną przy wykorzystaniu zjawiska fotowoltaicznego [14]. *Zjawisko fotowoltaiczne* polega na wzbudzaniu elektronów znajdujących się w kryształach krzemu, przez co zmieniają się ich poziomy energetyczne, a w efekcie następuje ich wybicie. Wybite elektrony znajdują się w paśmie przewodnictwa w półprzewodniku, co skutkuje przewartościowaniem materiału i ich uporządkowanym ruchem. Uporządkowany ruch elektronów to prąd elektryczny, w tym przypadku prąd stały. Jest on zamieniany na prąd przemienny przez *falownik*.

Zjawisko fotowoltaiczne jest podstawą działania *systemów fotowoltaicznych*. Przykładowy schemat systemu fotowoltaicznego pokazano na rysunku 1.1. Systemy te przetwarzają energię słoneczną w energię elektryczną, a w ich składzie wyróżnić można:

- *panele fotowoltaiczne* – produkujące prąd stały w oparciu o proces wybijania elektronów przez padające na warstwę półprzewodnika fotony (promienie słoneczne),
- *falownik* – przetwarzający prąd stały wyprodukowany przez panele w prąd przemienny,
- *instalację elektryczną* – zapewniającą przepływ prądu między elementami systemu.

Od kilku lat można zaobserwować lawinowy wzrost popularności systemów fotowoltaicznych. Niemały na to wpływ mają programy dofinansowania, dzięki którym użytkownicy mogą uzyskać środki na budowę takich instalacji. Aktualnie inwestycja w przydomowy system fotowoltaiczny zwraca się w okresie od 5 do 10 lat. Przy okazji jest to więc niezły sposób na lokowanie kapitału.

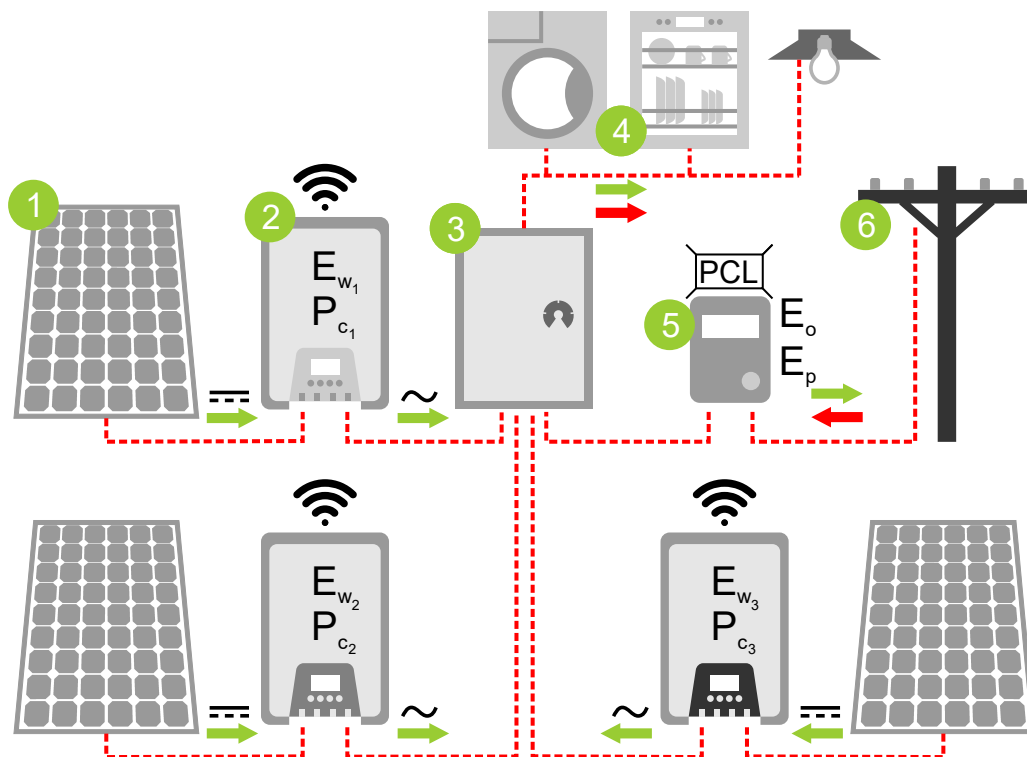
Uruchamiane źródła finansowania budowy instalacji fotowoltaicznych zwykle charakteryzowały się różnymi formalnymi warunkami do spełnienia. Warunki te, jak również aktualna sytuacja na rynku wpływały, m.in., na wybór producenta montowanej instalacji. Zdarzyło się więc, że klienci wchodzili w posiadanie kilku instalacji wyprodukowanych przez różnych producentów, do monitorowania których zaoferowano różne aplikacje.

Na rysunku 1.1 przedstawiono przykładowy schemat instalacji elektrycznej, w której funkcjonuje wiele systemów fotowoltaicznych. Zaznaczono na nim następujące elementy:

1. panel fotowoltaiczny,
2. falownik,
3. rozdzielna prądowa,
4. odbiorniki w gospodarstwie domowym,
5. publiczna sieć elektryczna,
6. dwukierunkowy licznik prądu.

Dodatkowo wprowadzono następujące oznaczenia:

- E_{w_i} – energia wyprodukowana przez instalację fotowoltaiczną nr i ,
- P_{c_i} – aktualnie produkowana moc przez instalację fotowoltaiczną nr i ,
- E_o – energia wprowadzona do publicznej sieci elektrycznej,
- E_p – energia pobrana z publicznej sieci elektrycznej.



Rys. 1.1: Schemat instalacji elektrycznej z trzema systemami fotowoltaicznymi

Klienci decydują się na inwestowanie w instalacje fotowoltaiczne, aby zmniejszyć koszty energii oraz przyczynić się do poprawy klimatu naszej planety. Z uwagi na panujące regulacje generowany prąd najlepiej jest zużyć na miejscu niż przekazywać go do publicznej sieci. Dzieje się tak, ponieważ otrzymywany zwrot za przekazaną energię jest mniejszy niż cena zakupu tej energii oferowana przez komercyjnych dostawców. Monitorowanie bieżącej produkcji może zatem pozwolić na optymalizację wykorzystania dostępnej energii. Strategia za tym stojąca polegać ma na uruchamianiu odbiorników prądu w chwilach, gdy własna produkcja na to pozwala. Dotyczy to w szczególności odbiorników, których godzina uruchomienia może być

ruchoma, na przykład: pralek, suszarek, zmywarek czy grzałek elektrycznych. Te ostatnie są dobrym przykładem, ponieważ mogą zastąpić emisyjne źródła ciepła, jak np. piece gazowe.

W przypadku, gdy użytkownicy posiadają wiele instalacji fotowoltaicznych, tak jak pokazano to na rysunku 1.1, monitorowanie ich działania jest utrudnione. Każdy producent dostarcza bowiem inną aplikację do kontrolowania zainstalowanego systemu. Inne interfejsy aplikacji oraz różne jednostki prezentowanych wartości skutecznie utrudniają użytkownikom panowanie nad sumarycznymi parametrami instalacji fotowoltaicznych.

Potrzeba jest matką wynalazków, dlatego powstał pomysł zbudowania systemu informatycznego agregującego dane z wielu instalacji elektrycznych oraz licznika energetycznego. W zamierzeniach miałby on umożliwiać monitorowanie bieżącego stanu produkcji energii elektrycznej oraz dawać wgląd w jej historię za pośrednictwem czytelnych i przyjaznych użytkownikowi widoków. Jego użycie powinno dać szansę na odpowiednie reagowanie na zmiany sumarycznej produkcji oraz pozwalać na planowanie. Użytkownicy takiego systemu mogliby efektywniej zarządzać produkowaną i zużywaną energią. Wgląd w aktualny stan posiadanych systemów fotowoltaicznych powinien przełożyć się na lepsze zagospodarowanie ich potencjału, a tym samym na zmniejszenie ilości energii pobieranej z sieci. To bez wątpienia może pozytywnie wpłynąć na poprawę stanu klimatu naszej planety.

1.2. Cel pracy

Celem pracy jest zaprojektowanie oraz zaimplementowanie systemu ułatwiającego monitorowanie działania przydomowych instalacji fotowoltaicznych. System ten powinien umożliwiać komunikację z falownikami oraz licznikiem energii elektrycznej celem pozyskania danych, pozwalać na agregowanie i przetwarzanie pozyskanych danych, dostarczać wyniki przetwarzania w przyjaznej dla użytkownika formie. Aplikacja ma być dostępna publicznie zabezpieczona przez moduł autoryzacji i uwierzytelnienia. System powinien być konfigurowalny i możliwy do wdrożenia bez konieczności technicznej ingerencji w instalację elektryczną użytkownika.

Aby wyróżnić projektowany system od innych systemów postanowiono nadać mu nazwę własną *SolarMonitoring*. Jako produkt system ten powinien dotrzeć do nietechnicznych odbiorców. Dlatego odpowiedni nacisk podczas jego implementacji ma być położony na przyjazność interfejsu oraz zapewnienie dobrego doświadczenia użytkownikom (ang. *User Experience*, UX). Użytkownicy powinni swobodnie odnajdywać się wśród oferowanych widoków. Wystawiane na nich funkcje mają być łatwo dostępne i nieskomplikowane w obsłudze oraz zapewniać realizację podstawowych wymagań (przeglądanie danych bieżących i historycznych, wizualizacja aktualnych stanów instalacji).

1.3. Zakres pracy

Budowany system składać się ma z dwóch głównych komponentów: aplikacji klienckiej oraz aplikacji serwerowej. Zatem zakres planowanych prac obejmie zaprojektowanie i zaimplementowanie obu tych aplikacji. Jest to dość rozległy zakres. Co więcej, sam proces wytwarzania oprogramowania może okazać się nietrywialny. Kluczowe fragmenty jego przebiegu powinny zostać odpowiednio udokumentowane.

Aplikacja serwerowa będzie odpowiadać za przetwarzanie danych dostarczanych z 3 źródeł:

- aplikacji klienckiej – dane wprowadzane przez użytkowników,
- serwisów chmurowych systemów fotowoltaicznych – dane bieżące i historyczne,
- panelu administracyjnego operatora energii elektrycznej – historyczne dane pomiarowe licznika energii elektrycznej.

Pozyskane dane mają być przechowywane w bazie danych. Dane wrażliwe klientów będą przechowywane w bazie danych w bezpiecznej formie (haszowane lub szyfrowane).

Aplikacja kliencka będzie aplikacją internetową:

- o intuicyjnym interfejsie zabezpieczonym mechanizmami uwierzytelnienia i autoryzacji
- umożliwiającą użytkownikom zarządzanie własnymi kontami, instalacjami fotowoltaicznymi, licznikami energii elektrycznej
- generującą zestawienia danych historycznych i bieżących umożliwiającymi ich analizę (na wykresach i tzw. dashboardzie).

Aplikacja serwerowa i kliencka mają komunikować się poprzez interfejs typu REST wystawiany przez aplikację serwerową. Dalsze szczegółowe wymagania mają zostać określone na etapie analizy wymagań.

W zakresie pracy znajduje się również zapoznanie się z aspektami, które nie były omawiane w toku studiów, w szczególności z wdrażaniem aplikacji w serwisach chmurowych.

1.4. Układ pracy

Rozdział 2

Projekt

2.1. Inicjalizacja projektu

Rozpoczynając pracę nad projektem podejmuje się wiele ważnych decyzji. Dotyczą one, m.in., implementowanych funkcji, przyjętego sposobu przechowywania kodu źródłowego, środowiska programistycznego oraz wdrożeniowego, docelowej architektury, wykorzystanych technologii itp. Decyzje te materializują się w formie formalnie zdefiniowanych wymagań.

Ponieważ SolarMonitoring z założenia miał być łatwo dostępny, zdecydowano się na budowę przyjaznego interfejsu użytkownika osiągalnego za pośrednictwem aplikacji internetowej. W konsekwencji decyzja ta narzuciła popularny wzorzec architektury systemu do wykorzystania, bazujący na wykorzystaniu schematu komunikacji typu klient-serwer. We wzorcu tym wyróżnia się dwa główne moduły: aplikację klienta oraz aplikację serwera. Zaplanowano, że oba moduły zostaną wdrożone w wybranym serwisie chmurowym.

Aplikacja serwerowa odpowiadać ma za przetwarzanie danych: pobieranie danych ze źródeł danych, przechowywanie ich w bazie danych oraz wysyłanie ich do aplikacji klienckiej. Pewnym standardem stało się budowanie takich aplikacji z wykorzystaniem konwencji REST API. Taką też konwencję postanowiono przyjąć w niniejszej pracy. W przypadku aplikacji klienckiej wybór technologii jest zależny od planowanego zakresu dostarczanych w niej funkcji. Analizę tych funkcji wraz z bazowymi technologiami opisano w dalszej części pracy.

Jeśli chodzi o sposób przechowywania kodów źródłowych aplikacji, to postanowiono skorzystać z repozytorium zapewniającego wersjonowanie kodu. Wybór padł na jedno z najpopularniejszych repozytoriów jakim jest Github.

2.2. Wymagania funkcjonalne

Przystępując do realizacji systemu SolarMonitoring zdefiniowano następującą listę funkcji, które zaoferować ma swojemu potencjalnemu użytkownikowi:

FR-1. Zarządzanie kontem użytkownika:

- FR-1.1. rejestracja użytkownika,
- FR-1.2. logowanie użytkownika,
- FR-1.3. edycja danych użytkownika,
- FR-1.4. usuwanie użytkownika,
- FR-1.5. wylogowywanie użytkownika.

FR-2. Zarządzanie falownikami:

- FR-2.1. dodawanie falownika,
- FR-2.2. usuwanie falownika,

FR-2.3. edycja danych falownika.

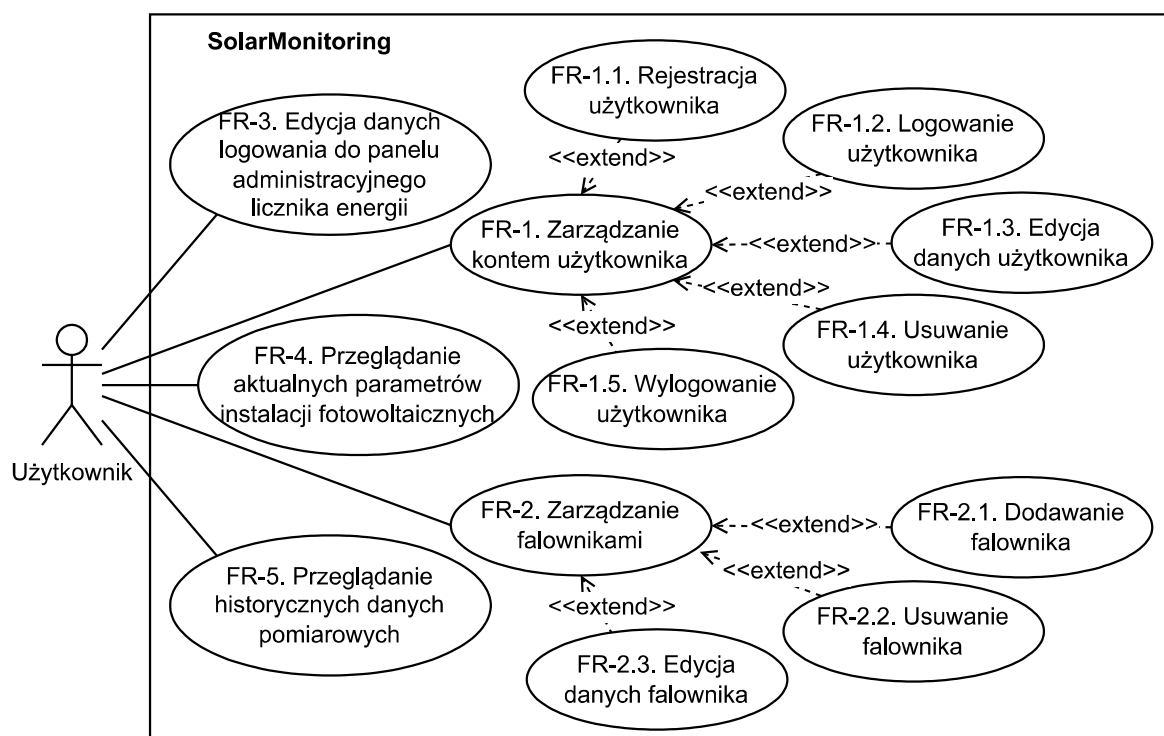
FR-3. Edycja danych logowania do panelu administracyjnego operatora energii elektrycznej

FR-4. Przeglądanie aktualnych parametrów instalacji fotowoltaicznych: stan, produkowana moc, produkcja bieżącego dnia, produkcja bieżącego roku.

FR-5. Przeglądanie historycznych danych pomiarowych z gradacją miesięczną i możliwością wyboru roku. Prezentowane dane:

- sumaryczna energia wyprodukowana (E_w),
- energia oddana do sieci (E_o),
- energia pobrana z sieci (E_p),
- energia wyprodukowana zużyta bezpośrednio w gospodarstwie domowym ($E_w - E_o$).

Z powyższej listy można wnioskować, że w systemie będzie istnieć tylko jedna rola: Użytkownik – osoba posiadający instalacje fotowoltaiczne. Będzie ona mogła korzystać z systemu bez konieczności działania innego typu użytkownika. Na rysunku 2.1 zilustrowano wymagania funkcjonalne w formie diagramu przypadków użycia systemu SolarMonitoring. Przypadki te uzupełnia opis zamieszczony dalej.



Rys. 2.1: Diagram przypadków użycia systemu SolarMonitoring

Opisy przypadków użycia

FR-1.1. Rejestracja użytkownika

Aktor: Użytkownik

Warunki wstępne: Brak warunków wstępnych.

Warunki końcowe: Użytkownik pomyślnie utworzył swoje konto.

Scenariusz główny:

1. Otworzyć aplikację kliencką w przeglądarce internetowej.
2. Kliknąć przycisk "Zarejestruj się".
3. Uzupełnić pola: imię, nazwisko, email, login, hasło, potwierdzenie hasła.
4. Kliknąć przycisk "Zarejestruj się".

5. Rejestracja przebiegła pomyślnie o czym użytkownik zostanie poinformowany odpowiednim komunikatem.

Scenariusz alternatywny:

3. Użytkownik niepoprawnie wypełnił formularz (nie uzupełnił pewnych pól lub hasło i jego potwierdzenie nie są zgodne) – użytkownik zostanie poinformowany odpowiednim komunikatem co należy poprawić.
4. Rejestracja może się nie powieść, ponieważ użytkownik o podanym loginie istnieje.

FR-1.2. Logowanie użytkownika

Aktor: Użytkownik

Warunki wstępne: Użytkownik posiada konto.

Warunki końcowe: Użytkownik pomyślnie się zalogował.

Scenariusz główny:

1. Otworzyć aplikację kliencką w przeglądarce internetowej.
2. Kliknąć przycisk "Zarejestruj się".
3. Uzupełnić pola: login, hasło.
4. Kliknąć przycisk "Zaloguj się".
5. Użytkownik zostanie przekierowany na stronę główną aplikacji.

Scenariusz alternatywny:

- 5a. Użytkownik niepoprawnie wypełnił formularz (nie uzupełnił pola login bądź hasła) – użytkownik zostanie poinformowany odpowiednim komunikatem co należy poprawić.
- 5b. Logowanie się nie powiodło, ponieważ użytkownik wprowadził niepoprawne dane.

FR-1.3. Edycja danych użytkownika

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany do aplikacji.

Warunki końcowe: Użytkownik pomyślnie zmienił swoje dane.

Scenariusz główny:

1. Otworzyć zakładkę "Użytkownik".
2. Zmienić interesujące dane użytkownika.
3. Kliknąć przycisk "Zapisz".
4. Po poprawnym zapisie użytkownik zostanie poinformowany odpowiednim komunikatem.

Scenariusz alternatywny:

4. Użytkownik niepoprawnie wypełnił formularz (nie uzupełnił pola login bądź hasła) – użytkownik zostanie poinformowany odpowiednim komunikatem co należy poprawić.

FR-1.4. Usuwanie użytkownika

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany do aplikacji.

Warunki końcowe: Konto użytkownika zostało usunięte.

Scenariusz główny:

1. Otworzyć zakładkę "Użytkownik".
2. Kliknąć przycisk "Usuń konto".
3. Użytkownik zostanie przekierowany na stronę logowania.

Scenariusz alternatywny: brak.

FR-1.5. Wylogowanie użytkownika

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany.

Warunki końcowe: Użytkownik pomyślnie się wylogował.

Scenariusz główny:

1. Kliknąć przycisk "Wyloguj się".

2. Użytkownik zostanie przekierowany na stronę logowania.
Scenariusz alternatywny: brak.

FR-2.1. Dodawanie falownika

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany i posiada dane dostępowe do API falownika.

Warunki końcowe: Falownik został dodany.

Scenariusz główny:

1. Otworzyć zakładkę "Dodaj falownik".
2. Wybrać odpowiednią markę falownika.
3. Wprowadzić dane falownika:
 - Solax: numer seryjny oraz token id,
 - SolarEdge: klucz API oraz site id.
4. Kliknąć przycisk "Zapisz".
5. Użytkownik zostanie przekierowany na stronę zawierającą listę falowników.

Scenariusz alternatywny:

5. Użytkownik niepoprawnie wypełnił formularz (nie uzupełnił wymaganych pól) – użytkownik zostanie poinformowany odpowiednim komunikatem co należy poprawić.

FR-2.2. Usuwanie falownika

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany.

Warunki końcowe: Falownik został usunięty.

Scenariusz główny:

1. Otworzyć zakładkę "Lista falowników".
2. W kontenerze odpowiedniego falownika kliknąć przycisk "Usuń".
3. Użytkownik zostanie poinformowany odpowiednim komunikatem o wyniku operacji.

Scenariusz alternatywny: brak.

2.3. Edycja danych falownika

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany.

Warunki końcowe: Dane falownika zostały zaktualizowane.

Scenariusz główny:

1. Otworzyć zakładkę "Lista falowników".
2. W kontenerze wybranego falownika kliknąć przycisk "Edytuj".
3. Wprowadzić dane falownika:
 - 3.1. Solax: numer seryjny oraz token id,
 - 3.2. SolarEdge: klucz API oraz site id.
4. Kliknąć przycisk "Zapisz".
5. Użytkownik zostanie przekierowany na stronę zawierającą listę falowników.

Scenariusz alternatywny:

5. Użytkownik niepoprawnie wypełnił formularz (nie uzupełnił wymaganych pól) – użytkownik zostanie poinformowany odpowiednim komunikatem co należy poprawić.

FR-3. Edycja danych logowania do panelu administracyjnego operatora energii elektrycznej

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany oraz posiada przygotowane dane logowania do panelu administracyjnego operatora energii elektrycznej.

Warunki końcowe: Dane logowania zostały zaktualizowane.

Scenariusz główny:

1. Otworzyć zakładkę "Licznik".
2. Wprowadzić dane: login, hasło.
3. Kliknąć przycisk "Zapisz".
4. Użytkownik zostanie poinformowany odpowiednim komunikatem o pomyślności wykonanej akcji.

Scenariusz alternatywny:

3. Użytkownik niepoprawnie wypełnił formularz (nie uzupełnił wymaganych pól) – użytkownik zostanie poinformowany odpowiednim komunikatem co należy poprawić.

FR-4. Przeglądanie aktualnych parametrów instalacji fotowoltaicznych

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany.

Warunki końcowe: Użytkownik odczytał bieżące parametry instalacji fotowoltaicznych.

Scenariusz główny:

1. Otworzyć zakładkę "Lista falowników".
2. Na liście falowników dla każdej z instalacji można odczytać następujące parametry:
 - stan instalacji fotowoltaicznej: czuwanie, normalna praca lub błąd połączenia,
 - bieżącą produkcję [W],
 - energię wyprodukowaną w danym dniu [kWh],
 - energię wyprodukowaną w danym roku [MWh].

Scenariusz alternatywny: brak.

FR-5. Przeglądanie historycznych danych pomiarowych z gradacją miesięczną i możliwością wyboru roku

Aktor: Użytkownik

Warunki wstępne: Użytkownik jest zalogowany.

Warunki końcowe: Użytkownik odczytał historyczne dane z wybranego roku.

Scenariusz główny:

1. Otworzyć zakładkę "Pomiary".
2. Wybrać interesujący rok.
3. Na wykresie liniowym, dla danego roku z gradacją miesięczną można odczytać następujące dane:
 - sumaryczna energia wyprodukowana (E_w),
 - energia oddana do sieci (E_o),
 - energia pobrana z sieci (E_p),
 - energia wyprodukowana zużyta bezpośrednio w gospodarstwie domowym ($E_w - E_o$).

Scenariusz alternatywny:

3. Użytkownik wprowadził rok, w którym nie były archiwizowane dane – zostanie poinformowany o tym odpowiednim komunikatem.

2.3. Wymagania niefunkcjonalne

Kolejnym etapem projektu było zdefiniowanie wymagań niefunkcjonalnych. Zasadniczym wymaganiem tego typu jest ogólnodostępność systemu – aplikacja powinna być dostępna dla wszystkich użytkowników sieci Internet. Ponadto wymagana jest niezawodność. Cecha ta jest w dużej mierze zależna od dostawcy serwisu chmurowego, na którym aplikacja zostanie wdrożona.

Ważnym aspektem jest też bezpieczeństwo danych użytkownika – poufne dane użytkownika powinny być przechowywane w bazie danych w zaszyfrowanej formie. W szczególności hasło użytkownika powinno być przechowywane jako jego funkcja skrótu – dzięki czemu można porównać wprowadzane hasło z oryginałem, jednak nie można poznać oryginalnego hasła. Ko-

munikacja między serwerem a aplikacją kliencką powinna być prowadzona szyfrowanym protokołem HTTPS, ponieważ te aplikacje komunikują się kanałem komunikacyjnym publicznej sieci Internet, który jest narażony na podsłuchy.

System powinien umożliwiać monitorowanie jego stanu przez administratorów. Należy zaimplementować odpowiedni sposób logowania informacji z działania systemu. Odpowiednie logowanie takich informacji ułatwi w przyszłości rozwiązywanie ewentualnych problemów.

2.4. Wymagania dziedzinowe

System ma zbierać, przetwarzać i przechowywać dane pomiarowe z różnych źródeł. Jak pokazano na rysunku 1.1 podstawowe parametry to:

- E_{w_i} – energia wyprodukowana przez instalację fotowoltaiczną nr i , jednostka: kWh ,
- P_{c_i} – aktualnie produkowana moc przez instalację fotowoltaiczną nr i , jednostka: W ,
- E_o – energia wprowadzona do publicznej sieci elektrycznej, jednostka: kWh ,
- E_p – energia pobrana z publicznej sieci elektrycznej, jednostka: kWh .

Na podstawie tych parametrów można obliczyć następujące parametry pochodne:

- E_w – sumaryczna energia wyprodukowana przez instalacje fotowoltaiczne, można ją obliczyć ze wzoru: $\sum_{i=1}^n E_{w_i}$,
- E_b – energia wyprodukowana zużyta bezpośrednio w gospodarstwie domowym, można ją obliczyć ze wzoru: $E_b = E_w - E_o$.

Wszystkie parametry energetyczne (oznaczone symbolami E) będą prezentowane na wykresach słupkowych z gradacją miesięczną i podziałem na lata. Parametr P_{c_i} będzie prezentowany w formie tekstowej na widoku z listą instalacji fotowoltaicznych.

Sposób rozliczania energii oddanej i pobranej z sieci elektrycznej też wymaga wyjaśnienia. Klienci nieposiadający instalacji fotowoltaicznych to konsumenci, którzy kupują energię elektryczną w całości od operatora energii elektrycznej. Natomiast klienci posiadający instalacje fotowoltaiczne to prosumenci rozliczają energię wprowadzoną i pobraną z sieci w dosyć skomplikowany sposób. Obowiązujący algorytm rozliczania zależy od operatora energetycznego oraz daty założenia instalacji fotowoltaicznej.

Jednym z najpopularniejszych sposobów rozliczania, wykorzystywanym również w przypadku instalacji posiadanych przez autora pracy, jest metoda oparta o *wirtualny magazyn energii elektrycznej*. Ilość energii wprowadzonej do publicznej sieci elektrycznej jest pomniejszona o 20% i trafia do wirtualnego magazynu energii. Klient może wykorzystać zgromadzoną energię elektryczną w ciągu roku od daty wprowadzenia energii do magazynu. Instalacje fotowoltaiczne produkują najwięcej energii w miesiącach letnich, stąd dzięki wirtualnemu magazynowi energii nadwyżka energetyczna wyprodukowana w sezonie letnim może zostać wykorzystana zimą.

2.5. Model danych

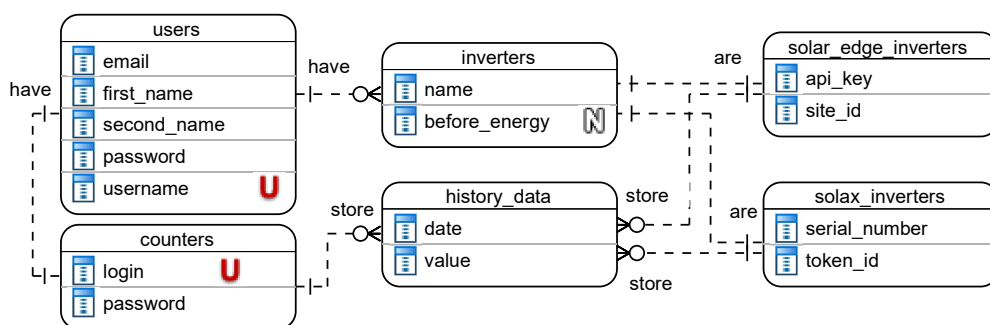
Mając na uwadze wymagania funkcjonalne można zdefiniować model danych. SolarMonitoring będzie przechowywał dane w bazie danych, zatem należy odpowiednio zdefiniować jej schemat. Zwykle w pierwszej kolejności definiowany jest *model konceptualny*, pozwalający usystematyzować koncepcje i wymagania biznesowe bez uwzględniania aspektów technicznych. Taki model przedstawia przede wszystkim dane przechowywane w encjach oraz zależności pomiędzy nimi. W kolejnym kroku definiowany jest *model fizyczny*. Dostarcza on technicznych szczegółów dotyczących sposobu przechowywania danych, w tym informacji o wykorzystywanych typach danych, sposobie realizacji związków między encjami oraz przyjętych więzach integralności [1].

Na rysunku 2.2 przedstawiono model konceptualny stworzony na podstawie wymagań funkcjonalnych dla systemu SolarMonitoring. Wyróżniono w nim następujące encje:

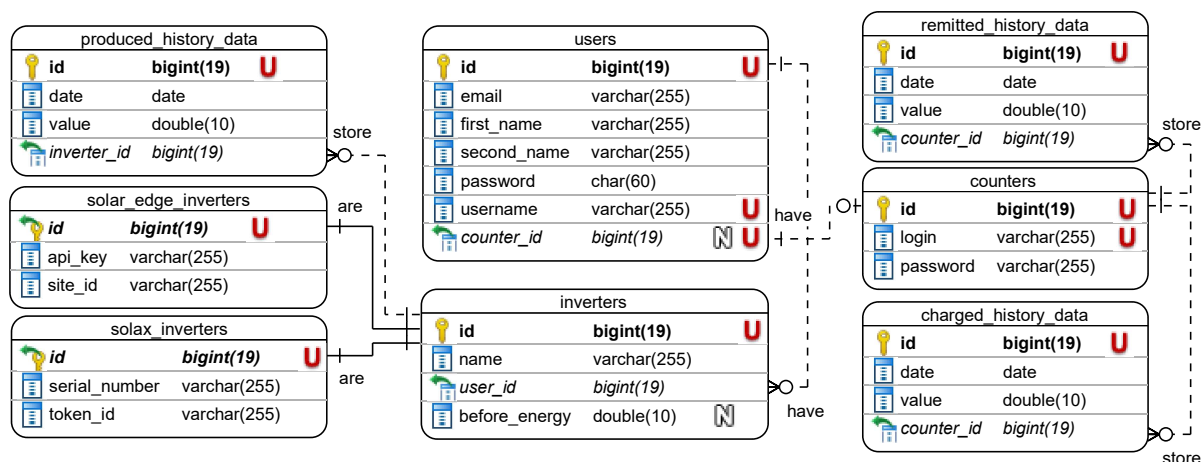
- **users** – encja reprezentująca użytkowników zapisanych w systemie. Zawiera podstawowe informacje o użytkownikach: imię, nazwisko, login, zahashowane hasło,
- **counters** – encja reprezentująca dane logowania do panelu administracyjnego Energa,
- **inverters** – encja przechowująca wspólne parametry dla każdej marki falowników: nazwę, sumaryczną energię zapisaną przy ostatnim odczycie danych (zostanie to opisane później),
- **solar_edge_inverters** – encja przechowująca dane dostępne dla falowników marki SolarEdge,
- **solax_inverters** – encja przechowująca dane dostępne dla falowników marki Solax,
- **history_data** – encja przechowująca zapisane dane historyczne.

Atrybuty poszczególnych encji nie będą tu szczegółowo omówione. Widać je na wspomnianym rysunku, a ich znaczenie wynika z nadanej nazwy w języku angielskim. Na podstawie modelu konceptualnego 2.2 można utworzyć bardziej szczegółowy model fizyczny. Pokazano go na rysunku 2.3. W modelu fizycznym uwzględniono szczegółowe informacje o typach danych oraz sposobach reprezentacji związków pomiędzy encjami w bazie danych. Zostały dodane kolumny z kluczami podstawowymi oraz kluczami obcymi. Encja **history_data** była powiązana z trzema innymi encjami, dlatego została rozdzielona na:

- **remitted_history_data** – encja na dane historyczne o energii oddanej do sieci E_o ,
- **charged_history_data** – encja na dane historyczne o energii pobranej z sieci E_p ,
- **produced_history_data** – encja na dane historyczne o wyprodukowanej energii E_w .



Rys. 2.2: Model konceptualny



Rys. 2.3: Model fizyczny

2.6. Obsługiwane źródła danych

W zakresie pracy zostały wymienione źródła danych, z których aplikacja serwerowa będzie pozyskiwać dane pomiarowe, przetwarzać je, archiwizować oraz przysyłać do aplikacji klienckiej. Aby móc testować tworzone oprogramowanie autor zdecydował się na wybór takich źródeł danych, do których ma dostęp. Aby spełnić wymagania sformułowane w celu pracy należy pobierać dane zarówno z systemów fotowoltaicznych jak i z licznika energii elektrycznej. Aplikacja będzie komunikować się z następującymi markami falowników: SolarEdge oraz Solax. Ponadto obsługiwanym licznikiem energetycznym, będzie licznik dostarczany przez operatora energetycznego Energa. Ze względu na ograniczenia czasowe, lista obsługiwanych producentów będzie ograniczona tylko do powyższych dwóch marek. Podczas implementacji autor będzie starał się zapewnić, aby system był łatwo rozszerzalny o obsługę falowników kolejnych producentów.

2.6.1. Licznik energii elektrycznej Energa

Dane z licznika energii elektrycznej udostępniane są przez firmę Energa. Energa to polski operator energetyczny z grupy Orlen obsługujący gospodarstwa domowe m. in. w południowej części Wielkopolski. Operator od kilku lat stosuje liczniki energii elektrycznej, które automatycznie wysyłają dane pomiarowe do centralnego systemu informatycznego operatora. Komunikacja odbywa się z wykorzystaniem technologii PCL, co pokazano na rysunku 1.1. Licznik ten mierzy energię dwukierunkowo: energię wprowadzoną do publicznej sieci elektrycznej E_o oraz energię pobraną z publicznej sieci elektrycznej E_p . Licznik został przedstawiony na rysunku 2.4. Niestety operator nie udostępnia publicznie interfejsu do wymiany danych, jedynie panel administracyjny w postaci aplikacji internetowej. Z tego powodu aplikacja serwerowa będzie symulować kliknięcia po stronie internetowej i odczytywać dane wyświetlane w przeglądarce. W tym celu zostanie wykorzystana biblioteka Selenium. Panel administracyjny jest zabezpieczony, każdy użytkownik może odczytywać swoje dane pomiarowe dopiero po uprzednim zalogowaniu się.



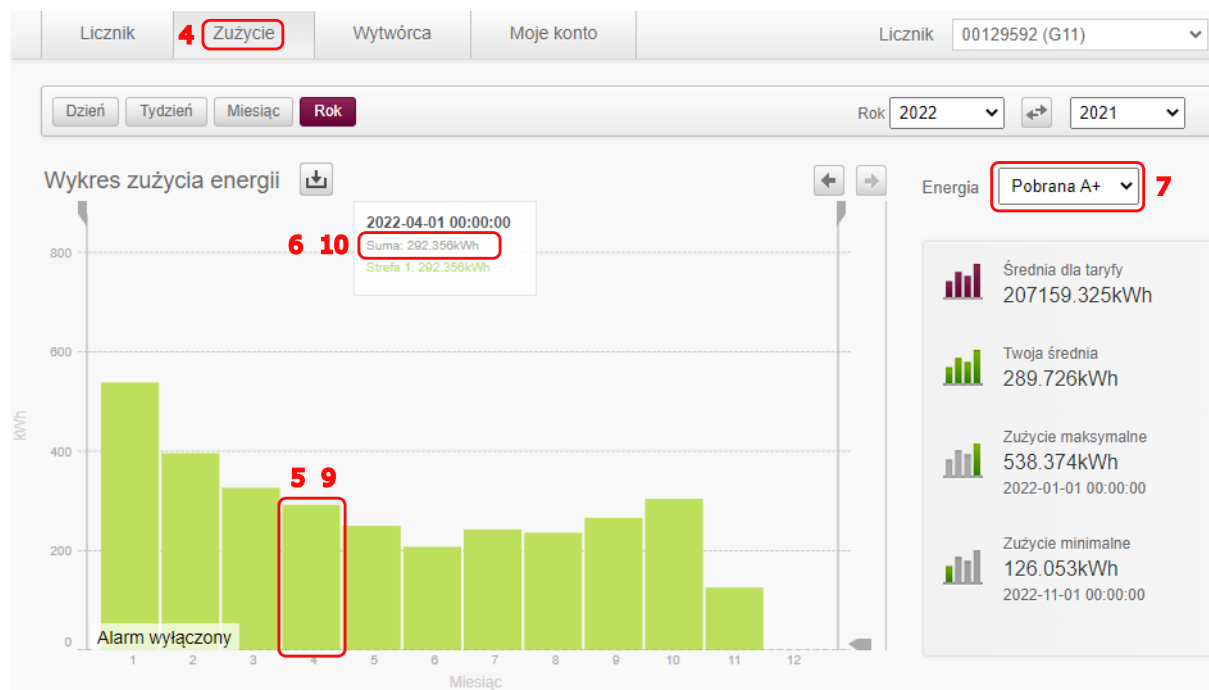
Rys. 2.4: Licznik energetyczny Energa – źródło: <https://blog-techniczny.pl/2020/12/13/monitorowanie-zuzycia-i-produkcji-pradu/>

Dane logowania powinny być zatem przechowywane przez aplikację serwerową w bazie danych, a z racji tego, iż są dane poufne, powinny być one przechowywane w zaszyfrowanej formie.

Panel administracyjny licznika energetycznego Energa jest dostępny pod adresem: <https://mojlicznik.energa-operator.pl/>. Dane tam prezentowane będą pozyskiwane za pośrednictwem biblioteki Selenium, która umożliwia oprogramowanie akcji użytkownika, które zwykle są wykonywane ręcznie na stronie internetowej. Widok strony logowania do omawianego panelu pokazano na rysunku 2.5, a pozyskiwanymi danymi będą:

- E_o – energia wyprodukowana wprowadzona do publicznej sieci elektrycznej (oddana) w wybranym miesiącu,
- E_p – energia pobrana z publicznej sieci elektrycznej.

Rys. 2.5: Strona logowania do panelu administracyjnego Energa



Rys. 2.6: Widok danych pomiarowych w panelu administracyjnym Energa

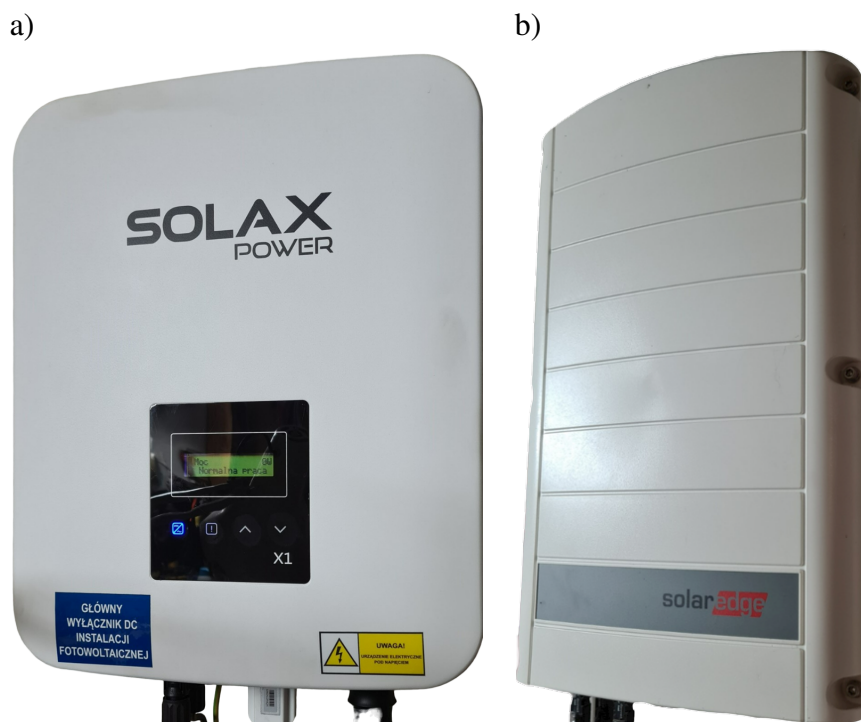
Należy jeszcze zdefiniować scenariusz interakcji ze stroną internetową związany z odczytaniem E_o oraz E_p . Symulowane akcje użytkownika będą odbywać się w następującej kolejności:

1. Wprowadzenie loginu.
2. Wprowadzenie hasła.
3. Klik na przycisku "Zaloguj się".
4. Klik na zakładce "Zużycie".
5. Najechnie kursorem na słupek odpowiedniego miesiąca.
6. Po pojawieniu się kontenera z wartościami odczytanie wartość pola "Suma". Jest to E_o w danym miesiącu.
7. W polu wyboru z podpisem "Energia" wybranie opcji "Oddana A-".
8. Poczekanie aż wykres się przeładuje.
9. Najechnie kursorem na słupek odpowiedniego miesiąca.
10. Po pojawieniu się kontenera z wartościami odczytanie wartość pola "Suma". Jest to E_p w danym miesiącu.

Uruchamianie akcji na graficznym interfejsie użytkownika zilustrowano na rysunkach 2.5 i 2.6 poprzez naniesienie liczb odpowiadających krokom scenariusza.

2.6.2. Falownik Solax

Jednym z producentów instalacji fotowoltaicznych, które będą obsługiwane przez SolarMonitoring jest Solax. Solax to producent budżetowej wersji systemów fotowoltaicznych. Falownik firmy Solax posiada moduł WiFi, który komunikuje z serwisem Solax Cloud. Solax Cloud udostępnia prosty interfejs API, z którego można pobierać dane bieżące takie jak aktualny stan systemu, aktualnie produkowaną moc, całkowitą produkcję itp. Zdjęcie falownika przedstawiono na rysunku 2.7a. W dokumentacji technicznej falownika zostało zdefiniowane API, z którego można korzystać za pośrednictwem sieci Internet. Aby móc pobierać dane należy znać parametry: numer seryjny modułu WiFi oraz identyfikator tokenu. Instrukcja pozyskania tych danych została opisana w dokumentacji technicznej falownika.



Rys. 2.7: Zdjęcia falowników: a) falownik firmy Solax, b) falownik firmy SolarEdge

Solax Cloud wystawia API z tylko jednym punktem końcowym (ang. *end-point*) [9]. Dane można uzyskać wywołując metodę HTTPS GET pod adresem: <https://www.solaxcloud.com:9443/proxy/api/getRealtimeInfo.do?tokenId={tokenId}&sn={sn}>, gdzie *tokenId* to identyfikator tokenu, a *sn* to numer seryjny modułu WiFi. Zapytanie zwraca dane w formacie JSON (ang. *Java Script Object Notation*). Ich szczegóły opisano w tabeli 2.1.

Tab. 2.1: Spis danych w odpowiedzi punktu końcowego *getRealtimeInfo* interfejsu Solax Cloud

Pole	Jednostka	Opis
inverterSN	–	Numer seryjny falownika
sn	–	Numer seryjny modułu WiFi
acpower	W	Produkowana moc
yieldtoday	kWh	Energia wyprodukowana w bieżącym dniu
yieldtotal	kWh	Energia wyprodukowana w całym okresie użytkowania
peps1	W	Moc fazy L1
peps2	W	Moc fazy L2
peps3	W	Moc fazy L3
inverterType	–	Typ falownika
inverterStatus	–	Status falownika
uploadTime	–	Czas ostatniej aktualizacji
batPower	W	Moc baterii
powerdc1	W	Moc MPPT1
powerdc2	W	Moc MPPT2

Nie wszystkie dane z tabeli 2.1 zostaną wykorzystane w aplikacji. Aplikacja serwerowa będzie wyłuskiwać z obiektu JSON następujące parametry: produkowaną moc (P_c), energię wyprodukowaną w bieżącym dniu, całkowitą wyprodukowaną energię oraz aktualny status.

2.6.3. Falownik SolarEdge

Kolejnym producentem obsługiwany przez SolarMonitoring jest SolarEdge. SolarEdge to producent systemów fotowoltaicznych wysokiej jakości z wieloma usprawnieniami technicznymi w szczególności optymalizatorami mocy, które zapobiegają ograniczaniu sumarycznej produkcji systemu przez niedoświetlone panele fotowoltaiczne. Falownik został przedstawiony na rysunku 2.7b. Falownik SolarEdge również jest wyposażony w moduł WiFi, za pośrednictwem którego przekazuje dane pomiarowe do serwisu SolarEdge Cloud. SolarEdge Cloud udostępnia zaawansowany interfejs API [8], z którym będzie komunikował się SolarMonitoring, jednak będzie on wykorzystany tylko w niewielkim stopniu.

Do komunikacji z API wymagane są następujące dane: klucz API (*apiKey*) oraz identyfikator strony (*siteId*). API SolarEdge Cloud jest bardziej rozbudowane niż API Solax Cloud i w tym przypadku wymagane będzie skorzystania z dwóch punktów końcowych: *overview* oraz *details*, które są dostępne pod adresami odpowiednio:

- https://monitoringapi.solaredge.com/site/{siteId}/overview?api_key={apiKey},
- https://monitoringapi.solaredge.com/site/{siteId}/details?api_key={apiKey}.

API SolarEdge Cloud również zwraca dane w formacie JSON. Zawartość odpowiedzi dla wymienionych punktów końcowych została opisana w tabelach 2.2 oraz 2.3. Podobnie jak w przypadku Solax'a, zostaną wykorzystane tylko następujące parametry: produkowana moc (P_c), energia wyprodukowana w bieżącym dniu, całkowita wyprodukowana energia oraz aktualny status.

Tab. 2.2: Spis danych w odpowiedzi punktu końcowego *overview* interfejsu SolarEdge Cloud

Pole	Jednostka	Opis
lastUpdateTime	–	Czas ostatniej aktualizacji
lifeTimeData	kWh	Energia wyprodukowana w całym okresie użytkowania
lastYearData	kWh	Energia wyprodukowana w bieżącym roku
lastMonthData	kWh	Energia wyprodukowana w bieżącym miesiącu
lastDayData	kWh	Energia wyprodukowana w bieżącym dniu
currentPower	W	Produkowana moc

Tab. 2.3: Spis danych w odpowiedzi punktu końcowego *details* interfejsu SolarEdge Cloud

Pole	Opis
id	Identyfikator instalacji
name	Nazwa instalacji
accountId	Identyfikator konta
status	Status instalacji
peakPower	Maksymalna produkowana moc
currency	Waluta
installationDate	Data oddania instalacji
ptoDate	Data udzielenia zezwolenia na użytkowanie instalacji
notes	Dodatkowe uwagi
type	Typ instalacji
location	Lokalizacja instalacji
alertQuantity	Liczba aktualnych ostrzeżeń
alertSeverity	Najwyższy poziom ostrzeżenia
publicSettings	Parametry dostępne publicznie

Rozdział 3

Implementacja

3.1. Architektura

W poprzednim rozdziale zostały wymienione wymagania co do systemu. Zostały zidentyfikowane encje, które system powinien przechowywać. Ponadto zostały opisane procedury pozyskiwania danych ze źródeł zewnętrznych. Na podstawie tych informacji można zdefiniować architekturę systemu SolarMonitoring, który będzie realizował określone wcześniej wymagania. System SolarMonitoring będzie składał się z następujących komponentów:

- *Serwer Spring* – aplikacja serwerowa pozyskująca dane z wymienionych wcześniej źródeł danych, przechowująca je w bazie danych oraz udostępniająca interfejs REST dla aplikacji klienckiej,
- *Baza danych* – relacyjny system bazodanowy przechowujący dane użytkowników,
- *Selenium Grid* – usługa komunikująca się z panelem administracyjnym Energa,
- *Serwer React* – serwer aplikacji klienckiej.

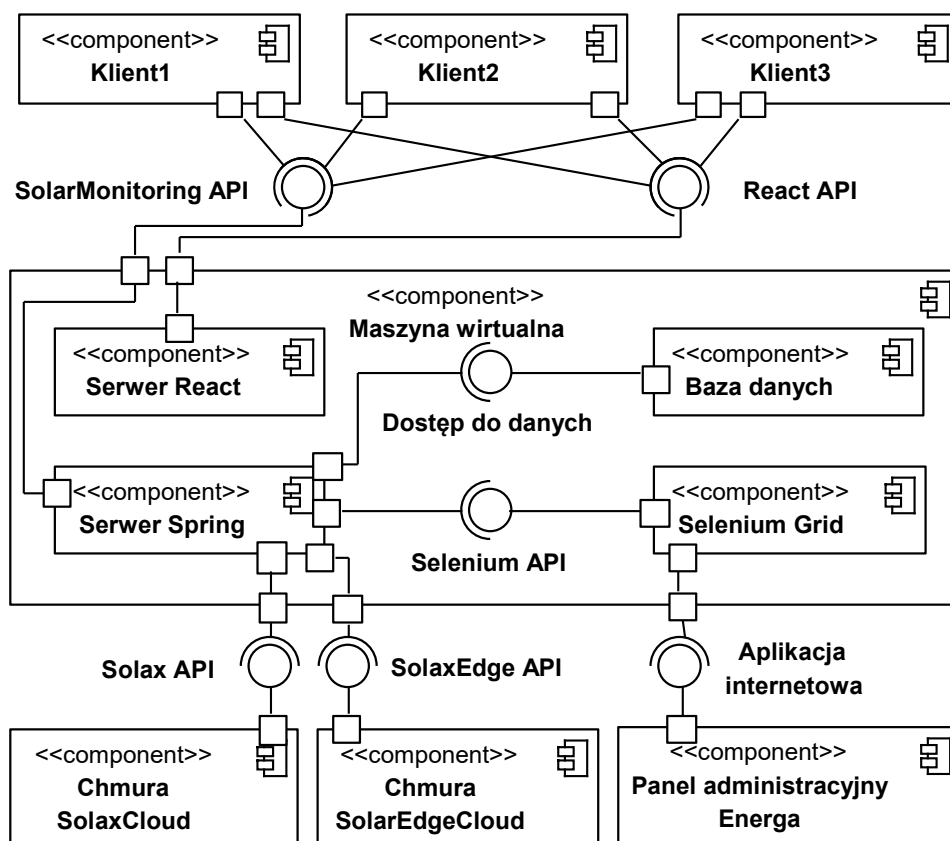
Powyższe komponenty to usługi, które będą uruchomione na jednej maszynie wirtualnej, dalej zwanej *serwerze*. Serwer będzie dostępny publicznie dla każdego użytkownika sieci Internet, dlatego docelowo wszystkie te komponenty zostaną wdrożone w wybranym serwisie chmurowym.

Wymienione komponenty oraz zależności pomiędzy nimi zostały przedstawione na schemacie architektury systemu zrealizowanym w formie diagramu komponentów na rysunku 3.1.

Na schemacie zostały zaznaczone dodatkowe komponenty:

- *Klient{1,2,3}* – przeglądarka internetowa użytkownika systemu komunikująca się z *serwerem React*,
- *Chmura SolaxCloud* – usługa chmurowa producenta SolaxCloud, udostępniająca interfejs opisany w sekcji 2.6.2,
- *Chmura SolarEdgeCloud* – usługa chmurowa producenta SolarEdgeCloud, udostępniająca interfejs opisany w sekcji 2.6.3,
- *Panel administracyjny energia* – aplikacja internetowa dostarczana przez operatora energetycznego Energa, z którą komunikuje się *Selenium Grid* w celu odczytu danych pomiarowych licznika energetycznego.

Schemat nie zawiera szczegółowych informacji na temat sposobów połączenia pomiędzy poszczególnymi komponentami, w szczególności sposób zabezpieczenia połączenia pomiędzy nimi, zostanie to opisane niżej.



Rys. 3.1: Architektura systemu SolarMonitoring

3.2. Aplikacja serwerowa

Aplikacja serwerowa odpowiedzialna jest za zarządzanie danymi. Przetwarza żądania aplikacji klienckiej i zwraca odpowiednie wyniki. Powinna udostępniać publiczny interfejs REST. Aplikacja serwerowa zarządza użytkownikami oraz powinna dostarczać funkcjonalności uwierzytelniania i autoryzacji. Powinna móc komunikować się z zewnętrznymi źródłami danych, w szczególności interfejsami usług chmurowych obsługiwanych falowników oraz aplikacją internetową panelu administracyjnego licznika energii elektrycznej.

Na podstawie wyżej zdefiniowanych wymagań co do aplikacji serwerowej można dokonać wyboru technologii, w której zostanie ona zaimplementowana. Do implementacji aplikacji serwerowej została wybrana biblioteka *Spring Boot*[15] oraz język programowania *Java*[7].

Spring Boot dostarcza wbudowane funkcje tworzenia REST API, komunikacji z bazą danych w oparciu o ORM oraz zabezpieczania komunikacji protokołem HTTPS. Z tego powodu implementacja aplikacji serwerowej powinna przebiegać sprawnie, zakładając iż autor pracy znał już wcześniej tę technologię.

3.2.1. Struktura projektu

Kod źródłowy aplikacji serwerowej był tworzony w odpowiedniej strukturze zgodnej z dobrymi praktykami programowania oraz praktykami narzucanymi przez bibliotekę *Spring*. Zostały wydzielone następujące pakiety:

- **config** – pakiet grupujący klasy konfiguracyjne aplikacji serwerowej, zawiera głównie klasy konfiguracyjne autoryzacji dla modułu *Spring Boot Security*, po za tym konfigurację przejrzysz-

stego panelu informacyjnego *Swagger UI* oraz konfigurację modułu logowania komunikatów programistycznych,

- **controller** – pakiet grupujący klasy kontrolerów, kontrolery w aplikacjach *Spring Boot* to klasy przetwarzające przychodzące żądania HTTP, czyli innymi słowy punkty wejściowe aplikacji.
- **custom** – mało znaczący pakiet, grupujący klasy związane z różnego rodzaju obejściami programistycznymi,
- **exchange** – pakiet grupujący klasy odpowiedzialne za pobieranie danych ze źródeł zewnętrznych: falowników oraz liczników energii elektrycznej,
- **model** – pakiet grupujący klasy modelujące encje przedstawione na rysunku 2.3, klasy reprezentujące zawartość zapytań i odpowiedzi serwera oraz klasy typu *mappers* mapujące klasy encji na odpowiedzi serwera lub zapytania na klasy encji,
- **repository** – pakiet grupujący interfejsy repozytoriów, repozytoria to interfejsy dostarczane przez moduł *Spring Data* służące do komunikacji z bazą danych,
- **scheduling** – pakiet grupujący klasy odpowiedzialne za automatyczne pobieranie danych ze źródeł danych zgodnie ze zdefiniowanym harmonogramem,
- **service** – pakiet grupujący klasy serwisów, czyli komponentów, które są odpowiedzialne za modelowanie logiki biznesowej systemu,

Dodatkowo w tym miejscu należy wspomnieć o dodatkowych folderach i plikach:

- plik `application.properties` – konfiguracja biblioteki *Spring Boot*,
- plik `keystore.p12` – plik zawierający parę kluczy RSA niezbędnych do prowadzenia szyfrowanej komunikacji z serwerem,
- folder `test` – folder zawierający testy aplikacji,
- pakiet `utils` w folderze `test` – pakiet grupujący klasy testowe z testami jednostkowymi aplikacji serwerowej,
- pakiet `web` w folderze `test` – pakiet grupujący klasy testowe z testami akceptacyjnymi aplikacji klienckiej napisane z wykorzystaniem biblioteki *Selenium*.

3.2.2. Najważniejsze fragmenty implementacji

Aplikacja serwerowa to głównie implementacja czyli modelowanie logiki biznesowej za pomocą wybranego języka programowania w tym przypadku *Javy*. W tej sekcji skupimy się na przytoczeniu najważniejszych fragmentów implementacji. Na listingu 3.7 została przedstawiona główna klasa aplikacji serwerowej, która uruchamia procesy startowe aplikacji *Spring Boot*, w szczególności wbudowany serwer *Apache Tomcat*.

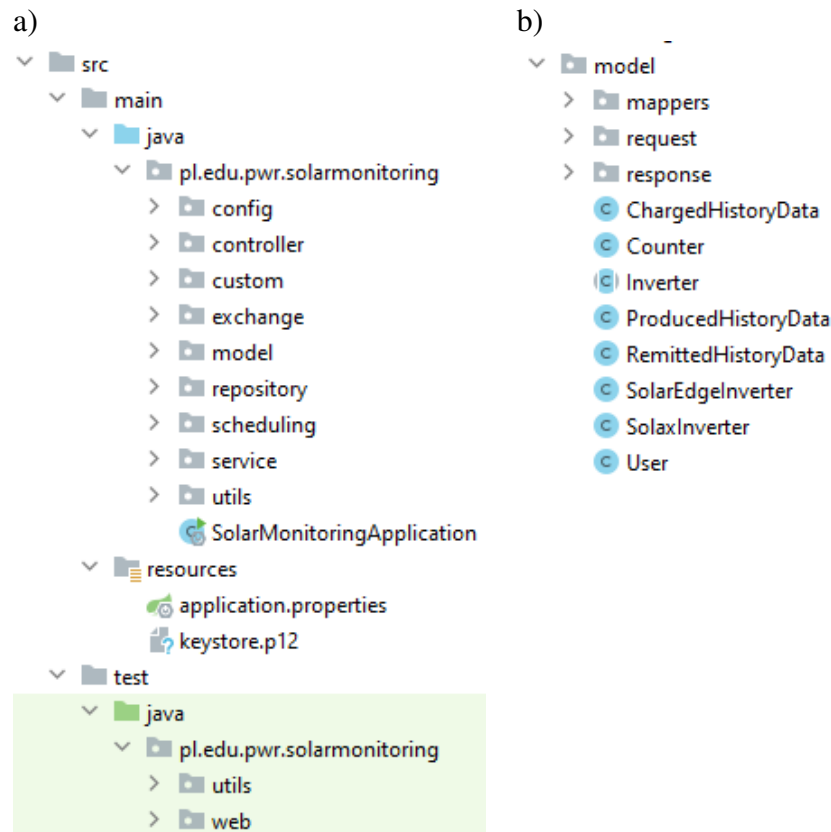
Listing 3.1: Główna klasa aplikacji serwerowej

```
@SpringBootApplication
@EnableScheduling
public class SolarMonitoringApplication {

    public static void main(String[] args) {
        SpringApplication.run(SolarMonitoringApplication.class, args);
    }
}
```

Kolejnym ważnym elementem jest plik konfiguracyjny aplikacji *Spring Boot* z podstawową konfiguracją serwera zapisaną w strukturze klucz-wartość. Najważniejsze parametry to:

- `spring.datasource.url` – adres serwera bazy danych,
- `spring.datasource.username` – nazwa użytkownika bazy danych,
- `spring.datasource.password` – hasło użytkownika bazy danych,



Rys. 3.2: Struktura projektu aplikacji serwerowej: a) struktura pakietów, b) zawartość pakietu *model*

- `spring.jpa.hibernate.ddl-auto` – parametr definiujący zachowanie systemu podczas napotkania różnicy pomiędzy schematem zaimplementowanym w bazie danych, a modelem danych zaimplementowanym w kodzie źródłowym aplikacji, wartość `update` oznacza, że w przypadku napotkania różnicy, schemat bazy danych zostanie zaktualizowany,
- `server.port` – numer portu aplikacji serwerowej,
- `server.ssl.enabled` – jeśli wartość `true` to komunikacja z serwerem będzie szyfrowana protokołem HTTPS,
- `server.ssl.key-store` – ścieżka do pliku z kluczami.

Listing 3.2: Plik konfiguracyjny aplikacji *Spring Boot*

```
spring.datasource.url=jdbc:postgresql://localhost:9286/
    ↪ solar_monitoring
spring.datasource.username=USERNAME
spring.datasource.password=PASSWORD
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
    ↪ PostgreSQL81Dialect
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false

logging.level.org.springframework.web.filter.
    ↪ CommonsRequestLoggingFilter=DEBUG

server.port=8000
server.ssl.enabled=true
```

```

server.ssl.key-store=classpath:keystore.p12
server.ssl.key-store-type=PKCS12
server.ssl.key-store-password=secret
server.ssl.key-alias=solarmonitoring
server.ssl.key-password=secret

logging.level.pl.edu.pwr.solarmonitoring=DEBUG

springdoc.api-docs.path=/api-docs

```

3.2.3. Zabezpieczenia

Aplikacja serwerowa zarządza i przechowuje dane poufne użytkowników, dlatego wymaga się od niej odpowiedniego poziomu zabezpieczeń. W aplikacji serwerowej można wyszczególnić 3 aspekty związane z zabezpieczeniem:

- szyfrowanie komunikacji z serwerem,
- zabezpieczanie danych przechowywanych w bazie danych,
- uwierzytelnianie i autoryzacja użytkowników.

Spring Boot dostarcza gotowej implementacji protokołu HTTPS, stąd po przygotowaniu odpowiedniej konfiguracji pokazanej na listingu 3.2 komunikacja z serwerem jest zabezpieczona szyfrowaniem niesymetrycznym z wykorzystaniem algorytmu RSA.

Kolejnym aspektem jest przechowanie poufnych danych przez aplikację serwerową. Można wyszczególnić następujące dane poufne:

- `User.password` – hasło w encji użytkownika, jest zapisane w bazie danych w postaci skrótu z wykorzystaniem funkcji *Bcrypt* [5], jest to szyfrowanie jednostronne, czyli na podstawie wartości zapisanej w bazie danych nie da się odzyskać oryginalnej wartości hasła, jedynie hasło wprowadzone przez użytkownika można porównać z hasłem zapisanym w bazie danych w celu sprawdzenia poprawności wprowadzonej wartości,
- `SolarEdgeInverter.apiKey` – klucz dostępowy do API *SolarEdge Cloud*, jest zapisane w bazie danych w postaci zaszyfrowanej z wykorzystaniem symetrycznego algorytmu szyfrowania *AES*,
- `SolaxInverter.tokenId` – klucz dostępowy do API *Solax Cloud*, jest zapisane w bazie danych w postaci zaszyfrowanej z wykorzystaniem symetrycznego algorytmu szyfrowania *AES*,
- `Counter.password` – hasło do panelu administracyjnego licznika energii elektrycznej Energa, jest zapisane w bazie danych w postaci zaszyfrowanej z wykorzystaniem symetrycznego algorytmu szyfrowania *AES*,

Listing 3.3 przedstawia klasę odpowiedzialną za szyfrowanie danych w aplikacji serwerowej. Metoda `encrypt` szyfruje dane algorytmem symetrycznym *AES*, które są potem zapisywane do bazy danych. Natomiast metoda `decrypt` odszyfrowuje dane odczytane z bazy danych. Ważne jest, aby podczas wywołania obu metod wartość klucza algorytmu *AES* `key` była taka sama. Obiekt `PASSWORD_ENCODER` jest odpowiedzialny za szyfrowanie jednostronne algorytmem *Bcrypt* poprzez wywołanie metody `encode` oraz sprawdzanie poprawności haseł poprzez wywołanie metody `matches`.

Listing 3.3: Klasa dostarczająca metod umożliwiających szyfrowanie

```

@Slf4j
public class EncryptionUtils {
    public static final BCryptPasswordEncoder PASSWORD_ENCODER = new
        ↪ BCryptPasswordEncoder();
    private static String key;

```

```

public static void setKey(String newKey) {
    key = newKey;
}
public static String encrypt(final String plainText) {
    try {
        byte[] decodedKey = Base64.getDecoder().decode(key);
        Cipher cipher = Cipher.getInstance("AES");
        SecretKey originalKey = new SecretKeySpec(Arrays.copyOf(
            ↪ decodedKey, 16), "AES");
        cipher.init(Cipher.ENCRYPT_MODE, originalKey);
        byte[] cipherText = cipher.doFinal(plainText.getBytes("UTF-8"))
            ↪ ;
        return Base64.getEncoder().encodeToString(cipherText);
    } catch (Exception e) {
        throw new RuntimeException("Error occured while encrypting
            ↪ plainText", e);
    }
}
public static String decrypt(final String encryptedString) {
    try {
        byte[] decodedKey = Base64.getDecoder().decode(key);
        Cipher cipher = Cipher.getInstance("AES");
        // rebuild key using SecretKeySpec
        SecretKey originalKey = new SecretKeySpec(Arrays.copyOf(
            ↪ decodedKey, 16), "AES");
        cipher.init(Cipher.DECRYPT_MODE, originalKey);
        byte[] cipherText = cipher.doFinal(Base64.getDecoder().decode(
            ↪ encryptedString));
        return new String(cipherText);
    } catch (Exception e) {
        throw new RuntimeException("Error occured while decrypting data
            ↪ ", e);
    }
}
}

```

Jeśli chodzi o uwierzytelnienie i autoryzację użytkowników, to w tym przypadku również przychodzi z pomocą biblioteka *Spring Boot*, która dostarcza moduł *Spring Security*. Zadaniem programistycznym było odpowiednie skonfigurowanie modułu. Najważniejszy fragment konfiguracji modułu *Spring Security* został przedstawiony na listingu 3.4.

Listing 3.4: Metoda configure z klasy WebSecurityConfig

```

protected void configure(HttpSecurity httpSecurity) throws Exception {
    CorsConfiguration corsConfiguration = new CorsConfiguration();
    corsConfiguration.setAllowedHeaders(List.of("Authorization", "Cache
        ↪ -Control", "Content-Type"));
    // dozwolone adresy, z których można komunikować się z serwerem
    corsConfiguration.setAllowedOrigins(List.of(crossOrigin1,
        ↪ crossOrigin2));
    // dozwolone metody HTTP
    corsConfiguration.setAllowedMethods(List.of("GET", "POST", "PUT", "
        ↪ DELETE", "PUT", "OPTIONS", "PATCH", "DELETE"));
    corsConfiguration.setAllowCredentials(true);
    corsConfiguration.setExposedHeaders(List.of("Authorization"));
    httpSecurity.cors().configurationSource(request ->
        ↪ corsConfiguration).and()
        .csrf().disable()
        // ścieżki, które nie wymagają autoryzacji
        .authorizeRequests().antMatchers(
            ↪ "/api/v1/user/authenticate",

```

```

        "/api/v1/user/create",
        "/api-docs/**",
        "/swagger-ui.html",
        "/swagger-ui/**"
    ).permitAll().
    anyRequest().authenticated().and().
    exceptionHandling().authenticationEntryPoint(
        ↪ jwtAuthenticationEntryPoint).and().sessionManagement
        ↪ ()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
// przekazanie obiektu, który dostarcza implementacji weryfikacji
↪ ważności tokenu JWT
httpSecurity.addFilterBefore(jwtRequestFilter,
    ↪ UsernamePasswordAuthenticationFilter.class);
}

```

Uwierzytelnienie odbywa się poprzez wywołanie w aplikacji klienckiej metody "POST" dla punktu wejściowego "api/v1/user/authenticate" z przekazaniem 2 parametrów: `username` oraz `password`. Aplikacja serwerowa weryfikuje wprowadzone dane, jeśli są poprawne odpowiada tokenem JWT (ang. *JSON Web Token*) [16], który jest zapisywany przez aplikację kliencką i używany później do autoryzacji.

3.2.4. Komunikacja ze źródłami danych

Jednym z podstawowych zadań aplikacji serwerowych jest pozyskiwanie danych ze źródeł danych. Wyróżniamy w aplikacji 3 źródła danych:

- licznik energii elektrycznej,
- falownik firmy SolarEdge,
- falownik firmy Solax.

Interfejsy udostępniane przez te urządzenia zostały opisane w sekcji 2.6, a poniżej zostanie przedstawiona implementacja pozyskiwania tych danych. Dane historyczne, które potem są archiwizowane w bazie danych są pozyskiwane zgodnie z ustalonym harmonogramem – dla falowników jest godzina 1:00 pierwszego dnia miesiąca, wtedy jest pobierana sumaryczna produkcja, od której odejmowana jest sumaryczna produkcja w poprzednim miesiącu (atrybut `before_energy` w tabeli `inverters` prezentowanej w modelu fizycznym 2.3). Wynikiem tego działania jest ilość energii wyprodukowanej w danym miesiącu.

W aplikacji *Spring Boot* w bardzo łatwy sposób można definiować zadania, które mają być uruchomione zgodnie z zadaniem harmonogramem. W tym celu należy do głównej klasy aplikacji dodać adnotację `@EnableScheduling`, a metody, które ma być uruchamiana automatycznie należy dodać adnotację `@Scheduled(cron = cronSchedule)`, gdzie `cronSchedule` to ciąg znaków definiujący harmonogram w składni *cron* [12]. Przykład metody, która jest uruchamiana automatycznie pierwszego dnia każdego miesiąca o godzinie 1:00 został przedstawiony na liście 3.5.

Metoda `saveMonthlyInvertersData` jest odpowiedzialna za archiwizowanie wyprodukowanej w poprzednim miesiącu energii przez falowniki. W pierwszej kolejności pobierani są wszyscy użytkownicy z bazy danych, następnie dla każdego falownika przypisanego do użytkownika jest odczytywana sumaryczna energia i obliczana jest energia wyprodukowana w poprzednim miesiącu, która jest na samym końcu zapisywana do bazy danych poprzez wywołanie `userRepository.saveAll(users)`, która zapisuje całą listę użytkowników wraz z danymi archiwalnymi, które są agregowane przez obiekty falowników, które z kolei są agregowane przez obiekty użytkowników.

Listing 3.5: Metoda `saveMonthlyInvertersData` uruchamiana zgodnie z zadaniem harmonogramem

```

@Scheduled(cron = "0 0 1 1 * *")
public void saveMonthlyInvertersData() {
    log.info("Started saveMonthlyInvertersData procedure");
    Collection<User> users = userRepository.findAll();
    for (User user : users) {
        saveMonthlyInvertersDataForUser(user);
    }
    userRepository.saveAll(users);
}

```

Licznik energii elektrycznej

Pobieranie danych z licznika energii elektrycznej jest realizowane zgodnie z harmonogramem – metoda archiwizująca jest uruchamiana 5. dnia każdego miesiąca o godzinie 0:00 (`cron = "0 0 0 5 * *"`). Są wtedy archiwizowane 2 wartości: E_o oraz E_p z poprzedniego miesiąca. Wartości te są pobierane z panelu administracyjnego licznika. Do pobierania danych ze strony internetowej zdecydowano się wykorzystać bibliotekę *Selenium*, która została stworzona w celu testowania aplikacji internetowych [2], jednak w tym przypadku zostanie wykorzystana do wchodzenia w interakcję ze stroną internetową panelu administracyjnego licznika przez aplikację serwerową.

Aplikacja serwerowa docelowo będzie wdrażana na serwerze. Serwery z reguły nie posiadają graficznego interfejsu użytkownika, więc nie będzie można na nich zainstalować jakiegokolwiek przeglądarki. Standardowym przypadkiem użycia biblioteki *Selenium* to skorzystanie z wybranego *WebDriver*'a. *Selenium* dostarcza następujących *WebDriver*'ów: *ChromeDriver*, *WebClient*, *SafariDriver*, *IOSDriver*, *AndroidDriver*, *FirefoxDriver*, *EdgeDriver*. Jednak służą one do wchodzenia w interakcję z przeglądarką internetową, która nie będzie dostępna na serwerze.

Z tego powodu należało zastosować inne podejście, należało skorzystać z *Selenium Grid*, który można zainstalować na dowolnym serwerze. Przy pomocy *RemoteWebDriver* dostarczanego przez bibliotekę *Selenium* oraz komunikując się z uruchomionym na tym samym serwerze *Selenium Grid'em*, można korzystać z funkcji dostarczanych przez bibliotekę *Selenium* na serwerze bez graficznego interfejsu użytkownika.

Falowniki

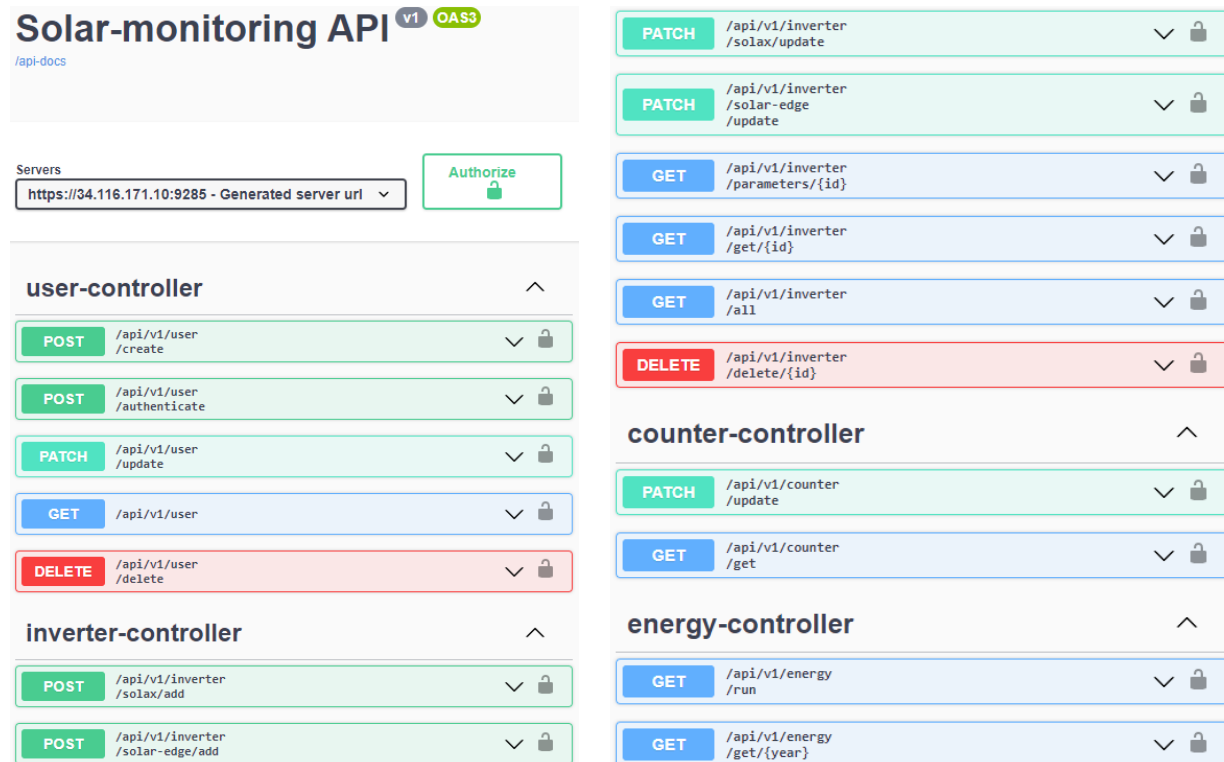
Dane z falowników pobierane na żądanie aplikacji klienckiej w momencie, w którym mają zostać wyświetlone aktualne parametry instalacji fotowoltaicznej oraz zgodnie ze zdefiniowanym harmonogramem. Metoda archiwizująca dane falowników jest uruchamiana automatycznie 1 dnia każdego miesiąca o godzinie 1:00 (`cron = "0 0 1 1 * *"`). Obie marki falowników udostępniają publiczny interfejs, z którym można komunikować się za pośrednictwem protokołu HTTP. Używane interfejsy zostały opisane w sekcjach 2.6.3 oraz 2.6.2. Do komunikacji HTTP został wykorzystany mechanizm dostarczany przez bibliotekę Spring - klasa *RestTemplate*. Wysłała ona żądanie HTTP i w odpowiedzi dostaje serializowany obiekt JSON, z którego później wyłuskiwane są oczekiwane parametry: status falownika, bieżąca moc, ilość energii wyprodukowanej danego dnia oraz sumaryczna energia wyprodukowana w całym okresie eksploatacji.

3.2.5. Struktura interfejsu REST

Aplikacja serwerowa z założenia udostępnia interfejs API, z którym będzie komunikować się aplikacja kliencka. Zdecydowano, że upubliczniany interfejs będzie spełniał wymagania standardyzowanego interfejsu typu REST. Implementacje punktów końcowych, tak jak już zostało

wspomniane wyżej, znajdują się w klasach z pakietu `controller`, aby klasy typu `controller` implementowały interfejs typu REST, należy stosować dla nich adnotację `@RestController`.

Aplikacja serwerowa nie posiada interfejsu użytkownika, z tego powodu zdecydowano się na wykorzystanie biblioteki *Swagger* [11], która automatycznie generuje dokumentację do udostępnianego przez aplikację API. Dzięki temu aplikacja serwerowa posiada komponent, który jest namacalny przez użytkownika. Dokumentacja ta została przedstawiona na rysunku 3.3.



Rys. 3.3: Struktura interfejsu REST

Punkty końcowe zostały podzielone w dokumentacji na kategorie – podział odpowiada kontrolerom, w których dane punkty końcowe są zaimplementowane. Aplikacja serwerowa posiada 4 kontrolery:

- **user-controller** – kontroler dostarczający metod do zarządzania użytkownikami,
- **inverter-controller** – kontroler dostarczający metod do zarządzania falownikami i odczytu ich bieżących parametrów,
- **counter-controller** – kontroler dostarczający metod do pobierania i edycji danych logowania do panelu administracyjnego licznika energii elektrycznej,
- **energy-controller** – kontroler dostarczający metod do odczytu danych archiwalnych.

Opis poszczególnych punktów końcowych został przedstawiony w tabeli 3.1. Ścieżki punktów końcowych nie zostały przedstawione w pełnej formie, każda ścieżka posiada ten sam przedrostek `/api/v1`, który w tabeli został pominięty.

3.3. Aplikacja kliencka

Aplikacja kliencka to aplikacja internetowa z interfejsem użytkownika. Celem aplikacji internetowej jest umożliwienie użytkownikowi korzystanie z systemu SolarMonitoring. Aplikacja kliencka jest tylko interfejsem, który wprowadzone dane przez użytkownika przekazuje do aplikacji serwerowej, a dane pozyskane z aplikacji serwerowej prezentuje użytkownikowi. Tak jak zostało to sformułowane w założeniach, aplikacja internetowa powinna posiadać przyjazny

Tab. 3.1: Opis API dostarczanego przez aplikację serwerową

Ścieżka	Metoda	Parametry	Opis
/user/create	POST	Dane użytkownika	Serwer tworzy nowego użytkownika. Jest to punkt końcowy niewymagający autoryzacji.
/user/authenticate	POST	Dane logowania	Serwer uwierzytelnia użytkownika na podstawie danych logowania. Jeśli uwierzytelnianie się powiodło zwraca token JWT. Jest to punkt końcowy niewymagający autoryzacji.
/user/update	PATCH	Zmienione dane użytkownika	Serwer aktualizuje dane użytkownika.
/user/	GET	–	Serwer zwraca dane użytkownika.
/user/delete	DELETE	–	Serwer usuwa danego użytkownika.
/inverter/solax/add	POST	Dane falownika Solax	Serwer tworzy nowy falownik marki Solax i przypisuje go do użytkownika.
/inverter/solar-edge/add	POST	Dane falownika SolarEdge	Serwer tworzy nowy falownik marki SolarEdge i przypisuje go do użytkownika.
/inverter/solax/update	PATCH	Zmienione dane falownika Solax	Serwer aktualizuje dane falownika firmy Solax.
/inverter/solar-edge/update	PATCH	Zmienione dane falownika SolarEdge	Serwer aktualizuje dane falownika firmy SolarEdge.
/inverter/parameters/{id}	GET	Identyfikator falownika	Serwer zwraca dane pomiarowe wybranego falownika.
/inverter/get/{id}	GET	Identyfikator falownika	Serwer zwraca atrybuty wybranego falownika.
/inverter/all	GET	–	Serwer zwraca listę wszystkich falowników wraz z danymi pomiarowymi oraz atrybutami.
/inverter/delete/{id}	DELETE	Identyfikator falownika	Serwer usuwa wybrany falownik.
/counter/update	PATCH	Dane logowania	Serwer aktualizuje dane logowania do panelu administracyjnego licznika energii elektrycznej przypisanego do użytkownika.
/counter/get	GET	–	Serwer zwraca login do panelu administracyjnego licznika energii elektrycznej przypisanego do użytkownika.
/energy/run	GET	–	Serwer uruchamia procedury archiwizacji danych pomiarowych.
/energy/get/{year}	GET	Rok	Serwer zwraca archiwalne dane pomiarowe.

dla użytkownika interfejs oraz zapewniać użytkownikom dobre doświadczenia podczas jej używania (ang. *User Experience*, UX). Biorąc pod uwagę powyższe założenia wybrano technologie, w których aplikacja kliencka będzie zaimplementowana.

React.js – jest biblioteką do tworzenia interfejsów webowych w języku JavaScript [10]. Został stworzony przez Facebook’a i aktualnie jest jedną z najpopularniejszych bibliotek tego typu. W React.js tworzone są komponenty, które mogą być używane w całej aplikacji. Raz zadeklarowany komponent może występować w wielu widokach aplikacji. Dodatkowo, React.js dostarcza mechanizm przyspieszający przeładowywanie komponentów w aplikacji internetowej co polepsza doświadczenia użytkownika. Mechanizm ten polega na tym, że podczas zmiany

struktury drzewa DOM strony internetowej, nie jest ona przeładowywana w całości, aktualizowana jest tylko część struktury, która wymaga zmiany. Mechanizm wygląda na dosyć prosty, jednak jego implementacja jest dosyć złożona, jednak zdecydowanie przyspiesza on interakcje użytkownika z aplikacją internetową.

Node.js – jest wieloplatformowym środowiskiem uruchomieniowym dla aplikacji serwerowych tworzonych w języku JavaScript [3]. Wybór tego środowiska był niejako narzucony przez wybór biblioteki *React.js* do tworzenia aplikacji klienckiej.

React-Bootstrap – jest biblioteką stosowaną przy budowie aplikacji implementowanych z wykorzystaniem React.js [6]. Dostarcza zbioru komponentów, które mogą być bezpośrednio używane w aplikacji. Główną zaletą tych komponentów jest ich ładny wygląd oraz responsywność (automatyczne dostosowywanie się do rozmiaru okna przeglądarki nawet małych rozmiarów, w szczególności przeglądarek uruchomionych na urządzeniach mobilnych).

Na podstawie wymagań funkcjonalnych, w szczególności listy przypadków użycia, można zdefiniować jakie widoki będą wymagane:

- W-1.** Logowanie
- W-2.** Rejestracja
- W-3.** O aplikacji
- W-4.** Pomiary
- W-5.** Licznik
- W-6.** Lista falowników
- W-7.** Dodawanie/edycja falownika
- W-8.** Użytkownik

W-1 Logowanie – rys. 3.5a

Widok który pozwoli użytkownikowi zalogować się do systemu. Głównym jego elementem będzie formularz zawierający 2 pola tekstowe: login oraz hasło. Oba pola są wymagane, zatem w przypadku jeśli użytkownik nie wprowadzi wartości, powinien zostać poinformowany odpowiednim komunikatem. Ponadto widok powinien zawierać odsyłacz do widoku rejestracji W-2. Po poprawnym logowaniu użytkownik powinien zostać przekierowany do widoku o aplikacji W-3.

W-2 Rejestracja – rys. 3.5b

Widok który pozwoli użytkownikowi zarejestrować się w systemie. Widok winien posiadać formularz zawierający następujące pola: imię, nazwisko, email, login, hasło oraz potwierdzenie hasła. Wszystkie pola są wymagane, zatem w przypadku jeśli użytkownik nie wprowadzi wartości, powinien zostać poinformowany odpowiednim komunikatem. Ponadto widok powinien zawierać odsyłacz do widoku logowania W-1. Po poprawnej rejestracji użytkownik powinien zostać poinformowany komunikatem.

Powyższe 2 widoki będą dostępne dla użytkowników niezalogowanych. Wszystkie pozostałe widoki, opisane niżej, będą posiadały pasek nawigacyjny, pozwalający na przełączanie się pomiędzy nimi oraz posiadający przycisk wylogowywania z systemu.

W-3 O aplikacji – rys. 3.6

Jest to widok opisujący aplikację. Zawierający nazwę aplikacji oraz listę producentów obsługiwanych falowników i licznika energii elektrycznej.

W-4 Pomiary – rys. 3.8

Jest to widok prezentujący na wykresie słupkowym historyczne dane pomiarowe archiwizowane przez system SolarMonitoring. Wykres powinien zawierać wartości dla wszystkich miesięcy wybranego roku. Widok będzie posiadał pole liczbowe, którym użytkownik będzie mógł wybrać rok, z którego dane chce odczytać. Na wykresie z gradacją miesięczną prezentowane będą następujące wartości (wszystkie wartości to ilość energii wyrażona w *kWh*): energia pobrana z sieci (E_p), energia oddana do sieci (E_o), sumaryczna produkcja (E_p), energia zużyta bezpośrednio ($E_p - E_o$), energia wyprodukowana przez poszczególne falowniki (E_{p_i}).

W-5 Licznik – rys. 3.10b

Jest to widok pozwalający użytkownikowi na aktualizację danych dostępowych do panelu administracyjnego licznika energii elektrycznej. Głównym elementem widoku powinien być formularz posiadający 3 pola: login, hasło oraz potwierdzenie hasła. Wszystkie pola są wymagane i w przypadku jeśli użytkownik nie wypełni jakiegokolwiek pola, powinien zostać poinformowany odpowiednim komunikatem.

W-6 List falowników – rys. 3.7

Jest to widok zawierający listę falowników wraz z ich bieżącymi parametrami. Komponent prezentujący dane falownika powinien posiadać odnośnik do widoku edycji jego danych dostępowych oraz przycisk usuwania. Przycisk usuwania powinien wyświetlać formularz potwierdzenia chęci usunięcia wybranego falownika z listy. Komponent falownika powinien wyświetlać następujące dane: nazwę, status, markę, bieżącą moc [*W*], dzisiejszą produkcję [*W*] oraz całkowitą produkcję [*kWh*].

W-7 Dodawanie/edycja falownika – rys. 3.9

Są to 2 widoki, które będą zaimplementowane w postaci tego samego komponentu. Jedyną różnicą będzie to, iż widok dodawania falownika będzie posiadał pusty formularz, natomiast widok edycji falownika będzie posiadał formularz wypełniony aktualnymi danymi. Formularz będzie posiadał następujące pola: nazwę, markę (pole wyboru z następującymi wartościami: *Solax* oraz *SolarEdge*) oraz w zależności od wybranej marki, dla marki *Solax*: numer seryjny oraz token id, a dla marki *SolarEdge*: api key oraz site id.

W-8 Użytkownik – rys. 3.10a

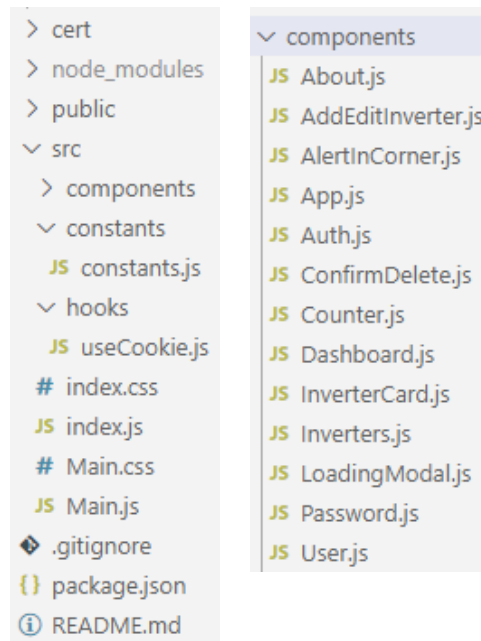
Jest to widok pozwalający użytkownikowi zmienić swoje dane. Głównym elementem widoku będzie formularz posiadający następujące pola: imię, nazwisko, email, login, stare hasło, nowe hasło oraz potwierdzenie nowego hasła. W przypadku chęci zmiany hasła, użytkownik powinien wprowadzić stare hasło oraz nowe wraz z jego potwierdzeniem. Pola wymagane bezwarunkowo to imię, nazwisko, email oraz login. Pole wymagane warunkowo to stare hasło, nowe hasło oraz jego potwierdzenie, w przypadku jeśli którekolwiek z nich jest wypełnione.

3.3.1. Struktura projektu

Kod źródłowy aplikacji klienckiej był tworzony w oparciu o standardową strukturę projektu proponowaną przez dokumentację dla biblioteki React.js. Projektując i implementując komponenty autor miał na uwadze to, aby były one reużywalne. Ponadto, podczas inicjalizacji projektu w bibliotece React.js wiele folderów zostało utworzonych automatycznie. Poniżej została przedstawiona lista poszczególnych plików i folderów prezentowanych na zrzucie ekranu 3.4:

- **cert** – folder zawierający 2 pliki: **cert.pem** oraz **key.pem**, klucz publiczny i prywatny do szyfrowania protokołem HTTPS komunikacji przeglądarek klientów z serwerem aplikacji mobilnej,

- `node_modules` – folder przechowujący zależności aplikacji klienckiej,
- `public` – folder przechowujący dane statyczne aplikacji, takie jak zdjęcia, pliki bibliotek, `index.html`, etc.,
- `src` – folder przechowujący pliki źródłowe aplikacji: kaskadowe arkusze stylów `*.css` oraz skrypty w języku JavaScript `*.js` definiujące działanie i logikę aplikacji oraz komponenty,
- `.gitignore` – plik systemu wersjonowania, definiujący, które elementy mają nie być objęte śledzeniem wersji,
- `package.json` – plik konfiguracyjny aplikacji *React.js* zawierający m. in. skrypty uruchomieniowe oraz listę zależności,
- `README.md` – plik zawierający opis jak uruchomić aplikację.



Rys. 3.4: Struktura projektu aplikacji klienckiej

Ponadto, na zrzucie ekranu 3.4 została przedstawiona lista komponentów zaimplementowanych w aplikacji klienckiej. Odpowiedzialności poszczególnych komponentów to:

- `About.js` – widok o aplikacji W-3,
- `AddEditInverter.js` – formularz dodawania falownika bądź edycji danych falownika zawarty w widoku W-7 – zależnie od przekazanego parametry,
- `AlertInCorner.js` – kontener wyświetlany w prawym górnym narożniku aplikacji z informacją zwrotną dla użytkownika po wykonaniu akcji, np. zapisu danych bądź rejestracji,
- `App.js` – punkt wejściowy aplikacji po zalogowaniu, posiadający definicję panelu nawigacyjnego oraz zarządzający, który widok powinien być wyświetlany,
- `Auth.js` – widoki logowania W-1 i rejestracji W-2,
- `ConfirmDelete.js` – komponent z panelem potwierdzającym usunięcie wybranego elementu (w przypadku systemu SolarMonitoring – użytkownika lub falownika),
- `Counter.js` – formularz edycji danych dostępowych do panelu administracyjnego licznika energii elektrycznej,
- `Dashboard.js` – widok pomiarów historycznych W-4,
- `InverterCard.js` – komponent odpowiedzialny za prezentację danych pojedynczego falownika w widoku listy falowników W-6,
- `Inverters.js` – widok listy falowników W-6,
- `LoadingModal.js` – wyskakujące okienko z obracającym się wycinkiem okręgu, który informuje użytkownika o ładowaniu danych,

- `Password.js` – komponent pola hasła z przyciskiem ukrywającym bądź wyświetlającym hasło,
- `User.js` – formularz edycji danych użytkownika zawarty w widoku W-8.

3.3.2. Najważniejsze fragmenty implementacji

Autor nie miał wcześniej do czynienia z technologią *React.js*, z tego powodu początkowo implementacja aplikacji klienckiej postępowała mało dynamicznie. Specyfika języka JavaScript oraz niektóre wzorce stosowane w bibliotece *React.js* są początkowo nieintuicyjne. Należało przede wszystkim zaznajomić się z dokumentacją i zaimplementować kilka podstawowych funkcjonalności, wtedy implementacja pozostałej części postępowała dosyć sprawnie. W tej sekcji zostaną przedstawione główne elementy implementacji oraz ciekawe przypadki, które autorowi przysporzyły niewielkich problemów.

Listing 3.6 przedstawia punkt wejściowy aplikacji internetowej, który renderuje dokładnie jeden komponent: `Main`. Kod źródłowy komponentu `Main` został przedstawiony na listingu 3.7.

Komponent `Main` jest odpowiedzialny za zarządzanie widokami w zależności od tego, czy dany użytkownik jest zalogowany czy też nie. Użytkownik zalogowany posiada zapisany token JWT w plikach HTTP cookies. Jeśli taki token nie jest zapisany tzn. że użytkownik nie jest zalogowany i powinien mieć dostęp jedynie do panelu logowania i rejestracji. W przypadku, w którym w pliku HTTP cookie jest zapisany token JWT, to oznacza, że taki użytkownik jest zalogowany. W takim przypadku, komponent `Main` zawiera komponent `App`, który jest głównym komponentem aplikacji dostępnym tylko i wyłącznie dla zalogowanych użytkowników.

Listing 3.6: Zawartość pliku `index.js`

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Main />
);
```

Listing 3.7: Kod źródłowy komponentu definiującego logikę związaną z wyborem odpowiedniego widoku w zależności od stanu uwierzytelnienia użytkownika

```
function Main() {

  function checkToken(text) {
    return typeof text === 'string' && text.length > 0
  }

  const [token, setToken] = useCookie("token", "");
  return (
    <BrowserRouter>
      {checkToken(token)===false &&
        <Auth setToken={setToken}/>
      }
      {checkToken(token)===true &&
        <App token={token} setToken={setToken}/>
      }
    </BrowserRouter>
  )
}

export default Main
```

Komponent `App` jest odpowiedzialny za zarządzanie widokami dostępnymi dla zalogowanych użytkowników. Posiada pasek nawigacyjny `Navbar` dostarczany przez bibliotekę *React-Bootstrap*. Pasek nawigacyjny składa się z linków do odpowiednich widoku oraz z przycisku wylogowywania. Do zarządzania widokami został wykorzystany *React Router* – moduł w bi-

bliotece *React.js* dedykowany do zarządzania wyświetlanymi komponentami w zależności od ścieżki URL aplikacji.

W komponencie *App* zostały wykorzystane ikony dostarczane przez bibliotekę *React Icons* dostępne na licencji otwartej, które poprawiają doświadczenia użytkowników korzystających z aplikacji. Dzięki ikonom obok przycisków czy linków aplikacja internetowa staje się bardziej intuicyjna.

Listing 3.8: Kod źródłowy głównego komponentu aplikacji po zalogowaniu – *App*

```
export default function App ({token, setToken}) {
  return (
    <div className="content">
      <Navbar collapseOnSelect expand="md" bg="primary" variant="dark">
        <Container>
          <Navbar.Brand href="about">Solar Monitoring</Navbar.Brand>
          <Navbar.Toggle aria-controls="responsive-navbar-nav" />
          <Navbar.Collapse id="responsive-navbar-nav">
            <Nav className="me-auto">
              <Nav.Link className='nav-item' href="/dashboard">
                Pomiary <AiOutlineHistory/>
              </Nav.Link>
              <Nav.Link className='nav-item' href="/counter">
                Licznik <BsFillCalculatorFill/>
              </Nav.Link>
              <NavDropdown id="nav-dropdown" title="Falowniki">
                <NavDropdown.Item href="/inverters">
                  Lista falowników <AiOutlineUnorderedList/>
                </NavDropdown.Item>
                <NavDropdown.Item href="/inverter">
                  Dodaj falownik <AiFillFolderAdd/>
                </NavDropdown.Item>
              </NavDropdown>
              <Nav.Link className='nav-item' href="/user">
                Użytkownik <AiOutlineUser/>
              </Nav.Link>
            </Nav>
            <Nav>
              <Nav.Link className='nav-item' onClick={() => {setToken("")}}>
                Wyloguj <MdOutlineLogout/>
              </Nav.Link>
            </Nav>
          </Navbar.Collapse>
        </Container>
      </Navbar>
      <Routes>
        <Route exact path="/"
          element={}<About/>{} />
        <Route path="/about"
          element={}<About/>{} />
        <Route path="/user"
          element={}<User token={token} setToken={setToken}/>{} />
        <Route path="/dashboard"
          element={}<Dashboard token={token}/>{} />
        <Route path="/inverters"
          element={}<Inverters token={token}/>{} />
        <Route path="/inverter"
          element={}<AddEditInverter token={token}/>{} />
        <Route path="/inverter/:id"
          element={}<AddEditInverter token={token}/>{} />
        <Route path="/counter">
```

```

        element={<Counter token={token}/>} />
      </Routes>
    </div>
  )}

```

Listing 3.9 zawiera kod źródłowy przykładowego komponentu, w tym przypadku jest to kod źródłowy komponentu `AlertInCorner`. Jest to obudowany, skonfigurowany i wzbogacony o odpowiednie style komponent `Alert` dostarczany przez bibliotekę *React Bootstrap*. Dodatkowe style zapewniają poprawne pozycjonowanie alertu na stronie internetowej. Ten komponent jest używany w wielu miejscach aplikacji, głównie do wyświetlania komunikatów związanych z komunikacją z aplikacją serwerową.

Listing 3.9: Kod źródłowy komponentu `AlertInCorner`

```

import Alert from 'react-bootstrap/Alert';
export default function AlertInCorner (props) {
  return (
    <div className="col-sm-6 AlertInCorner">
      <Alert variant={props.variant} onClose={props.onClose} dismissible
        ↪ style={{zIndex:1}}>
        {props.text}
      </Alert>
    </div>
  )}

```

Listing 3.10 zawiera kod źródłowy funkcji, która pobiera dane z aplikacji serwerowej systemu *SolarMonitoring*. Do komunikacji z serwerem została wykorzystana funkcja `fetch` dostarczana wraz z biblioteką *React.js*. Komunikacja z serwerem odbywa się w wielu miejscach aplikacji klienckiej, jednak w skorzystano z podobnego szablonu. Funkcja `fetch` może realizować wszystkie metody HTTP. Można w niej definiować nagłówki (ang. *headers*) oraz ciało (ang. *body*) zapytań HTTP. W tym przypadku wywołany jest punkt wejściowy aplikacji serwerowej dostępny pod ścieżką: `api/v1/user/authenticate`, który nie wymaga uwierzytelnienia. Metoda w ciele zapytania HTTP wysyła login oraz hasło użytkownika, a w odpowiedzi oczekuje tokenu JWT.

Listing 3.10: Funkcja `loginUser` pobierająca z serwera token JWT

```

function loginUser(credentials) {
  return fetch(Constants.SERVER_URL + 'user/authenticate', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(credentials)
  })
  .then((response) => {
    if (response.ok) {
      return response.json();
    }
    return Promise.reject(response);
  })
  .catch((response) => {
    console.log(response)
    setLoginFailed(true)
    setLoading(false)
  });
}

```

Aplikacja kliencka powinna wyświetlać wykresy z danymi historycznymi. W tym celu skorzystano z biblioteki *apexcharts* kompatybilnej z biblioteką *React.js* [13]. Listing 3.11 przed-

stawia fragment implementacji wykresu. Zmienna `series` zawiera słownik, gdzie kluczem są nazwy serii, natomiast wartościami historyczne dane pomiarowe. Zapis `{variable && <Component />}` oznacza, że `Component` będzie renderowany jeśli zmienna `variable` nie będzie `null` bądź `false`.

Listing 3.11: Fragment pliku `Dashboard.js` odpowiedzialny za renderowanie wykresu

```
<Card.Body>
  {series && <ReactApexChart options={options} series={series} type="bar"
    ↪ height={350} />}
</Card.Body>
```

Nie zostanie tutaj opisany każdy aspekt implementacji aplikacji klienckiej. Powyżej zostały opisane tylko jej kluczowe fragmenty. Pozostałe komponenty były tworzone w oparciu o zaprezentowane wyżej szablony komunikacji oraz sposobu implementacji komponentów z wykorzystaniem bibliotek *React.js* oraz *React Bootstrap*.

3.3.3. Instrukcja użytkownika

Posługując się zrzutami ekranu prototypu aplikacji klienckiej systemu SolarMonitoring została przedstawiona instrukcja użytkownika. Pierwszym widokiem aplikacji klienckiej jest widok logowania W-1 przedstawiony na rysunku 3.5a. Jeśli użytkownik posiada konto, może wprowadzić tutaj swoje dane logowania i kliknąć przycisk "Zaloguj". Jeśli wprowadzone dane logowania były poprawne, użytkownik zostanie przekierowany do widoku o aplikacji W-3. W przeciwnym wypadku zostanie poinformowany odpowiednim komunikatem podobnym do komunikatu przedstawionego na rysunku 3.12a.

Jeśli użytkownik nie posiada konta, to powinien kliknąć w odnośnik "Zarejestruj się". Zostanie wtedy przekierowany do widoku rejestracji W-2 przedstawionym na rysunku 3.5b. Wypełniając ten formularz oraz klikając przycisk "Zarejestruj się", użytkownik może utworzyć sobie swoje nowe konto. Ważne jest, aby wszystkie dostępne pola były wypełnione i ponadto pola "hasło" oraz "potwierdź hasło" posiadały taką samą wartość. Jeśli rejestracja przebiegła pomyślnie użytkownik zostanie poinformowany odpowiednim komunikatem. Użytkownik posiadający nowe konto powinien przez z powrotem do widoku logowania W-1 poprzez kliknięcie przycisku "Zaloguj się".

Pierwszym widokiem po zalogowaniu, jest widok o aplikacji W-3 przedstawiony na rysunku 3.6. Zawiera on podstawowe informacje na temat systemu SolarMonitoring.

Po zalogowaniu wszystkie widoki posiadają pasek nawigacyjny. Pasek nawigacyjny posiada następujące odnośniki:

- "Pomiary" – odnośnik przekierowujący użytkownika do widoku pomiarów W-4,
- "Licznik" – odnośnik przekierowujący użytkownika do widoku edycji danych dostępowych licznika W-5,
- "Falowniki" – rozwijane menu zawierające 2 odnośniki: "Lista falowników" – odnośnik przekierowujący użytkownika do widoku listy falowników W-6 oraz "Dodaj falownik" – odnośnik przekierowujący użytkownika do widoku dodawania falownika W-7, rozwijane menu zostało przedstawione na rysunku 3.12b,
- "Użytkownik" – odnośnik przekierowujący użytkownika do widoku edycji danych użytkownika W-8.

Klikając w odnośnik "Falowniki", a następnie w odnośnik "Lista falowników" użytkownik może przejść do widoku listy falowników W-6. Widok ten został przedstawiony na rysunku 3.7. Na przykładowym zrzucie ekranu znajdują się 3 przypisane do zalogowanego użytkownika falowniki: "Solax", "SolarEdge" oraz "Testowy falownik". Pierwsze 2 to pracujące falowniki, których wyświetlane parametry to rzeczywiste dane pomiarowe. Natomiast falownik "Testowy

a)

Zaloguj się

Nie posiadasz konta? [Zarejestruj się](#)

Login:

Hasło:

☐

Zaloguj się

b)

Zarejestruj się

Posiadasz już konto? [Zaloguj się](#)

Imię:

Nazwisko:

Email:

Login:

Hasło:

☐

Potwierdź hasło:

☐

Zarejestruj się

Rys. 3.5: Formularze: a) logowania W-1, b) rejestracji W-2

Solar Monitoring Pomiary ☺ Licznik 📊 Falowniki ▾ Użytkownik 👤 Wyloguj ↵

O aplikacji

Solar Monitoring
Aplikacja do monitorowania systemów fotowoltaicznych

Aplikacja pozwala na agregowanie danych z wielu systemów fotowoltaicznych. System komunikuje się z dwiema markami falowników: **Solax** i **SolarEdge**. System pobiera dane z licznika energetycznego. Licznik obsługiwany przez system to licznik dostarczany przez operatora **Energa**.

Rys. 3.6: Widok "W-3 O aplikacji"

falownik" został dodany poprzez wprowadzenie nieprawidłowych danych dostępowych z tego powodu jego status to "COMMUNICATION_ERROR", ponieważ aplikacja serwerowa nie mogła połączyć się z jego API.

Klikając w odnośnik "Pomiary" użytkownik może przejść do widoku pomiarów W-4. Widok ten został przedstawiony na rysunku 3.8. Widok ten umożliwia przeglądanie historycznych danych pomiarowych z gradacją miesięczną i podziałem na lata. Prezentowane dane to dane historyczne z roku 2022. Dane prezentowane są na wykresie słupkowym, każda wartość jest oznaczona innym kolorem, a pod wykresem znajduje się legenda z opisami poszczególnych kolorów.

Kontener każdego falownika posiada 2 przyciski: usuwania (czerowny) oraz edycji (niebieski). Przycisk usuwania powoduje wyświetlenie komunikatu potwierdzenia chęci usunięcia wybranego falownika. Komunikat ten jest podobny do komunikatu przedstawionego na rysunku

Solar Monitoring		Pomiary	Licznik	Falowniki	Użytkownik	Wyloguj
Solax						
Status: STANDBY						
Marka: Solax						
Bieżąca moc: 0 W						
Dzisiejsza produkcja: 0 kWh						
Całkowita produkcja: 1668 kWh						
Testowy falownik						
Status: COMMUNICATION_ERROR						
Marka: Solax						
Bieżąca moc: ---						
Dzisiejsza produkcja: ---						
Całkowita produkcja: ---						
SolarEdge						
Status: ACTIVE						
Marka: SolarEdge						
Bieżąca moc: 0 W						
Dzisiejsza produkcja: 0.77 kWh						
Całkowita produkcja: 23390.96 kWh						

Rys. 3.7: Widok "W-6 Lista falowników"

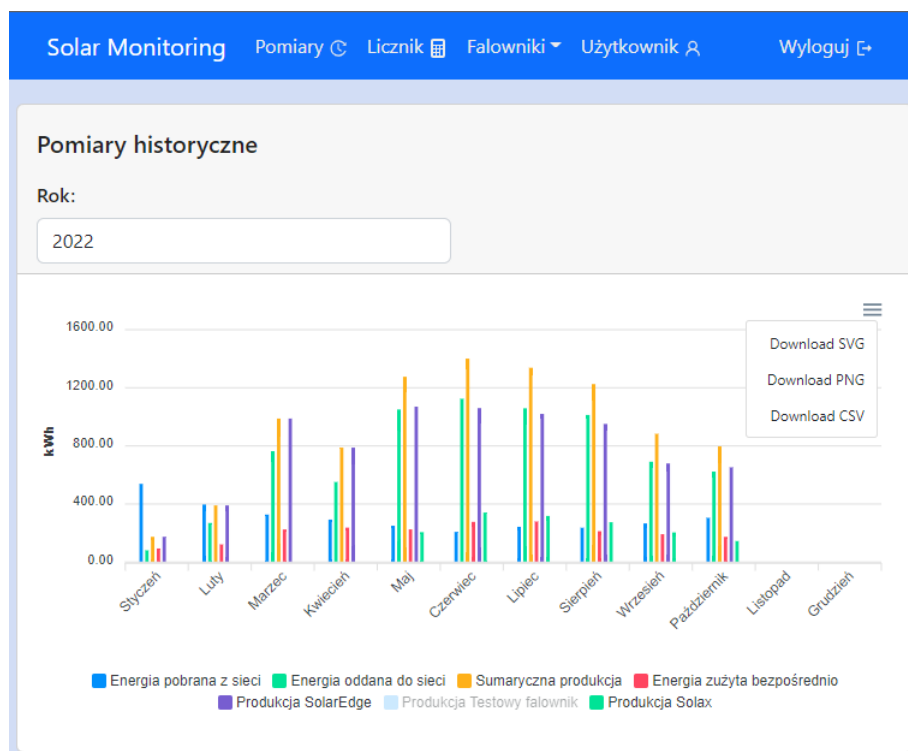
3.11. Przycisk edycji przekierowuje użytkownika do widoku edycji danych falownika W-7, który jest wypełniony bieżącymi danymi falownika i został przedstawiony na rysunku 3.9.

Istnieje możliwość filtrowania danych poprzez kliknięcie wybranej pozycji w legendzie – wtedy można ukrywać bądź wyświetlać wybraną serię danych. Ponadto biblioteka *apexcharts* dostarcza funkcjonalności eksportowania danych. Na rysunku 3.8 widać rozwinięte menu z następującymi opcjami: "Download SVG", "Download PNG" oraz "Download CSV".

Klikając w odnośnik "Falowniki", a następnie w odnośnik "Dodaj falownik" użytkownik może przejść do widoku dodawania falownika W-7. Widok ten został przedstawiony na rysunku 3.9. Głównym elementem widoku jest formularz "Dodaj falownik". Użytkownik może wprowadzić nazwę nowego falownika, wybrać jego markę oraz wprowadzić odpowiednie dla wybranej marki dane dostępowe. Po kliknięciu przycisku "Dodaj" aplikacja kliencka wysyła zapytanie do serwera, jeśli proces dodawania nowego falownika przebiegł pomyślnie, użytkownik zostanie poinformowany stosownym komunikatem podobnym do komunikatu przedstawionego na rysunku 3.12a.

Klikając w odnośnik "Licznik" użytkownik może przejść do widoku edycji danych dostępnych do panelu administracyjnego licznika energii elektrycznej W-5. Widok ten został przedstawiony na rysunku 3.10b. Widok składa się z formularza, który zawiera 3 pola: "login", "hasło" oraz "potwierdzenie hasła". Wszystkie pola są wymagane oraz pole "hasło" oraz "potwierdzenie hasła" muszą mieć tę samą wartość.

Klikając w odnośnik "Użytkownik" użytkownik może przejść do widoku edycji swoich danych W-8. Widok ten został przedstawiony na rysunku 3.10a. Widok składa się z formularza, który zawiera 7 pól. Wszystkie pola są wymagane oprócz pól związanych z hasłem użytkownika, które użytkownik wypełnia w przypadku jeśli chce zmienić swoje hasło. Po kliknięciu przycisku "Zapisz" zaktualizowane dane użytkownika są wysyłane do aplikacji serwerowej, która w odpowiedzi zwraca status wykonania operacji. Użytkownik o wyniku operacji zapisu jest informowany komunikatem podobnym do komunikatu przedstawionego na rysunku 3.12a.



Rys. 3.8: Widok "W-4 Pomiary"

Solar Monitoring

Dodaj falownik

Nazwa: Nowy falownik

Marka: Solax

Numer seryjny: SDF123SAD

Token id: da0ss97awl123asmr2tenc

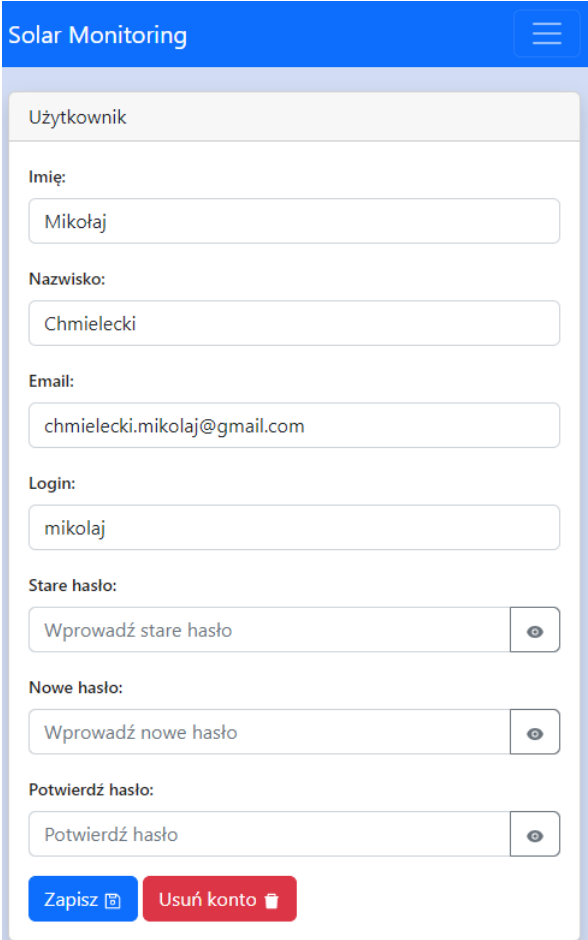
Dodaj

Rys. 3.9: Widok "W-7 Dodawanie/edycja falownika"

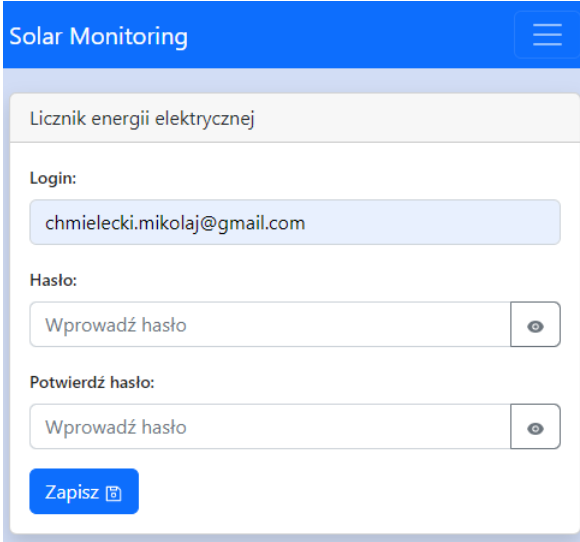
Ponadto, widok edycji danych użytkownika W-8 zawiera przycisk "Usuń konto", który powoduje wyświetlenie komunikatu potwierdzenia chęci usunięcia przedstawionego na rysunku 3.11. Jeśli użytkownik potwierdzi chęć usunięcia konta, konto zostanie usunięte z systemu, a użytkownik zostanie przekierowany na stronę logowania W-1.

Zrzuty ekranów formularzy 3.9, 3.10a oraz 3.10b zostały wykonane w bardzo zawężonym oknie przeglądarki w celu zaprezentowania responsywności aplikacji klienckiej. Widać, że pa-

a)

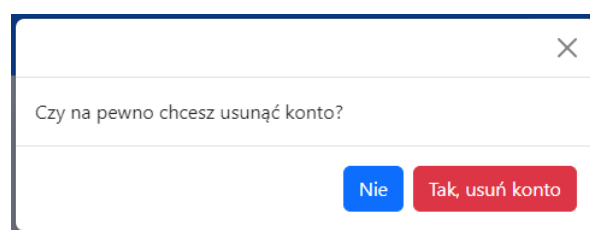


b)

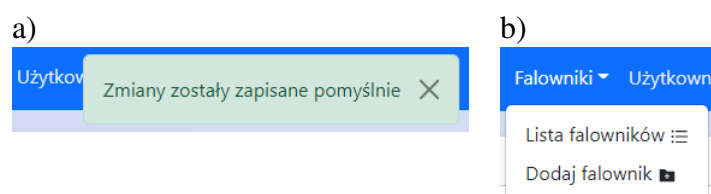


Rys. 3.10: Formularze: a) zmiany danych użytkownika W-8, b) zmiany danych logowania do panelu administracyjnego licznika W-5

sek nawigacyjny został ukryty. Jest on rozwijany i wyświetlany w postaci listy po kliknięciu w przycisk umieszczony w prawym górnym rogu ekranu.



Rys. 3.11: Komunikat potwierdzenia usunięcia



Rys. 3.12: Komponenty: a) komunikat użytkownika, b) rozwijane menu dla falowników

Rozdział 4

Testy aplikacji

4.1. Wykorzystane technologie

4.2. Testy jednostkowe

4.3. Testy automatyczne aplikacji internetowej

Rozdział 5

Podsumowanie

5.1. Plany rozwoju aplikacji

Literatura

- [1] S. Allen. *Modelowanie danych*. Wydawnictwo Helion, 2006.
- [2] B. Garcia. *Hands-On Selenium Webdriver with Java: A Deep Dive Into the Development of End-To-End Tests*. O'Reilly Media, 2022.
- [3] T. Hunter. *Distributed Systems with Node.js: Building Enterprise-Ready Backend Services*. O'Reilly UK Ltd., 2020.
- [4] T. Młynarski. Energetyka jądrowa wobec wyzwań zmian klimatu w XXI w. *Politeja*, 2016, Vol.13 (43), p.167-180, 2016.
- [5] N. Provos, D. Mazieres. A Future-Adaptable Password Schem. *The OpenBSD Proje*, 1999.
- [6] React-Bootstrap Documentation. <https://react-bootstrap.github.io/getting-started/introduction/>. Odczyt: 2022-11-30.
- [7] H. Schildt. *Java. Przewodnik dla początkujących*. Wydawnictwo Helion, 2020.
- [8] SolarEdge Monitoring Server API. https://www.eu.solaxcloud.com/phoebus/resource/files/userGuide/Solax_API_for_End-user_V1.0.pdf. Odczyt: 2022-11-28.
- [9] SolaX Cloud API for End-user. https://www.eu.solaxcloud.com/phoebus/resource/files/userGuide/Solax_API_for_End-user_V1.0.pdf. Odczyt: 2022-11-28.
- [10] S. Stefanov. *React: Up & Running, 2nd Edition*. O'Reilly Media, Inc., 2021.
- [11] Swagger UI – przejrzysta wizualizacja zasobów API. <https://bykowski.pl/swagger-ui-przejrzysta-wizualizacja-zasobow-api/>. Odczyt: 2022-11-28.
- [12] Understanding Cron Syntax in the Job Scheduler. <https://www.netiq.com/documentation/cloud-manager-2-5/ncm-reference/data/bexyssf.html>. Odczyt: 2022-11-28.
- [13] Using ApexCharts to create charts in React.js. <https://apexcharts.com/docs/react-charts/>. Odczyt: 2022-01-12.
- [14] Efekt fotowoltaiczny – co to jest? Definicja i zasada działania. <https://vosti.pl/aktualnosci/efekt-fotowoltaiczny-definicja-zasada-dzialania/>. Odczyt: 2022-11-02.
- [15] C. Walls. *Spring w akcji*. Wydawnictwo Helion, 2022.
- [16] What is JSON Web Token? <https://jwt.io/introduction>. Odczyt: 2022-11-27.

Dodatek A

Instrukcja wdrożeniowa

A.1. Wdrożenie aplikacji serwerowej wraz z bazą danych i serwisem Selenium Grid

Wdrażając aplikację serwerową, skorzystano z mechanizmu konteneryzacji dostarczanego przez narzędzie *docker*. Płyta DVD załączona do pracy zawiera obraz dockerowy aplikacji serwerowej, który można z łatwością uruchomić na serwerze. Wymagania wstępne:

- serwer z dowolnym systemem operacyjnym z rodziny Linux,
- dostęp do Internetu z publicznym adresem IP,
- odblokowany ruch TCP na porcie 9285,
- zainstalowane i skonfigurowane środowisko *OpenJDK* w wersji 11,
- zainstalowane i skonfigurowane narzędzie *Maven*,
- zainstalowane narzędzie *Docker*,
- zainstalowane narzędzie *docker-compose*.

W celu uruchomienia aplikacji serwerowej należy wykonać następujące kroki:

1. Z załączonej płyty DVD należy skopiować następujące pliki do wybranego folderu w masowej serwerze: `docker-compose.yml` oraz `image.tar`.
2. W wybranym folderze uruchomić następujące komendy:
 `> sudo dockerd`
 `> sudo docker load -i image.tar`
 `> sudo docker-compose up -d`
3. Po uruchomieniu powyższych komend należy sprawdzić czy serwisy zostały uruchomione poprawnie (serwisy backend, database oraz selenium powinny być w stanie health)
 `> sudo docker-compose ps`
4. Po uruchomieniu tych 3 serwisów należy sprawdzić czy dostępna jest dokumentacja *Swagger* dla wystawianego API przez aplikację serwerową w tym celu należy na innym urządzeniu w przeglądarce otworzyć następującą stronę: `https://<SERVER_IP>:9285/swagger-ui/index.html#/.` Jeśli strona się załadowała to znaczy, że aplikacja serwerowa została uruchomiona poprawnie.

A.2. Wdrożenie aplikacji klienckiej

Do uruchomienia aplikacji klienckiej należy spełnić następujące warunki wstępne:

- serwer z dowolnym systemem operacyjnym z rodziny Linux i dostępem do Internetu z publicznym adresem IP,

- odblokowany ruch TCP na porcie 3000,
- zainstalowane i skonfigurowane środowisko *Node.JS*.

W celu uruchomienia aplikacji klienckiej należy wykonać następujące kroki:

1. Z załączonej płyty DVD należy skopiować folder `src/frontend` do wybranego folderu w pamięci masowej serwera.
2. Przejść do skopiowanego folderu.
3. W pliku `src/constants/constants.js` zmienić wartość stałej `SERVER_URL` na wartość: `https://<SERVER_IP>:9285/api/v1/`, gdzie `SERVER_IP` to adres IP serwera.
4. Uruchomić następujące polecenie:
> `sudo npm start`
5. Po chwili powinien pojawić się komunikat `Webpack compiled successfully` – oznacza to, że serwer aplikacji klienckiej został uruchomiony poprawnie.

Aby sprawdzić poprawne działanie aplikacji klienckiej należy na innym urządzeniu uruchomić w przeglądarce stronę internetową: `https://<SERVER_IP>:3000/`. Jeżeli serwer aplikacji klienckiej działa poprawnie to w przeglądarce powinien pojawić się panel logowania.

Dodatek B

Opis załączonej płyty DVD

Załączona płyta DVD zawiera następujące elementy:

- `dyplom.pdf` – dokumentacja pracy inżynierskiej,
- `src/frontend/` – kod źródłowy aplikacji klienckiej,
- `src/backend/` – kod źródłowy aplikacji serwerowej,
- `image/backend.tar` – obraz dockerowy aplikacji serwerowej,
- `image/docker-compose.yml` – konfiguracja uruchomieniowa narzędzia *docker-compose* dla aplikacji serwerowej i bazy danych,
- `selenium/docker-compose.yml` – konfiguracja uruchomieniowa narzędzia *docker-compose* dla serwisu *Selenium Grid*.

Korzystając z instrukcji wdrożeniowej opisanej w dodatku A można uruchomić aplikację SolarMonitoring na przygotowanym wcześniej serwerze. W tym celu należy zgrać na pamięć masową serwera zbudowany obraz dockerowy aplikacji serwerowej i jego konfigurację oraz kod źródłowy aplikacji klienckiej.