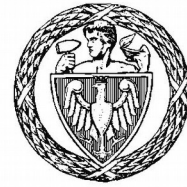


Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy Informacyjno-Decyzyjne

Zarządzanie ładowaniem pojazdów elektrycznych z wykorzystaniem
aukcji uwzględniającej mobilność

Michał Mokrogulski

Numer albumu 271227

promotor
dr inż. Izabela Żółtowska

Warszawa 2019

Zarządzanie ładowaniem pojazdów elektrycznych z wykorzystaniem aukcji uwzględniającej mobilność.

Streszczenie

Zarządzanie ładowaniem pojazdów elektrycznych opiera się na jednostce agregatora, który reprezentuje flotę pojazdów elektrycznych na rynku elektroenergetycznym. Agreguje on preferencje pojedynczych samochodów elektrycznych tak, aby były one znaczące na rynku, oraz zakupuje on potrzebną dla nich energię elektryczną.

W niniejszej pracy stworzono aplikację internetową, która umożliwia składanie deklaracji dostępności i odbieranie planów ładowania przez pojazdy elektryczne. Aplikacja umożliwia zintegrowanie, zarządzanie i monitorowanie procesu agregującego przez administratorów systemu. Aplikacja została wykonana w technologiach z otwartymi źródłami a jej zasadniczymi frameworki są Python, Django i Docker.

Aplikacja umożliwia podmiotom działającym na rynku zintegrowanie się z procesem agregującym i rozpoczęcie planowania ładowań samochodów elektrycznych.

Słowa kluczowe: inteligentne ładowanie pojazdów elektrycznych, agregator, aplikacja

Managing the charging of electric vehicles using auction including mobility.

Summary

Managing the charging of electric vehicles is based on the aggregator unit, which represents the fleet of electric vehicles on the electricity market. It aggregates the preferences of individual electric vehicles so that they are significant on the market, and it purchases the electricity they need for them.

In this work, an internet application has been created that enables making availability declarations and receiving charging plans for electric vehicles. The application enables integration, management and monitoring of the aggregation process by system administrators. The application has been made in open source technologies and its leading frameworks are Python, Django and Docker.

The application enables users to integrate with the aggregation process and start planning the charging of electric vehicles.

Keywords: intelligent charging of electric vehicles, aggregator, application



„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta”

Spis treści

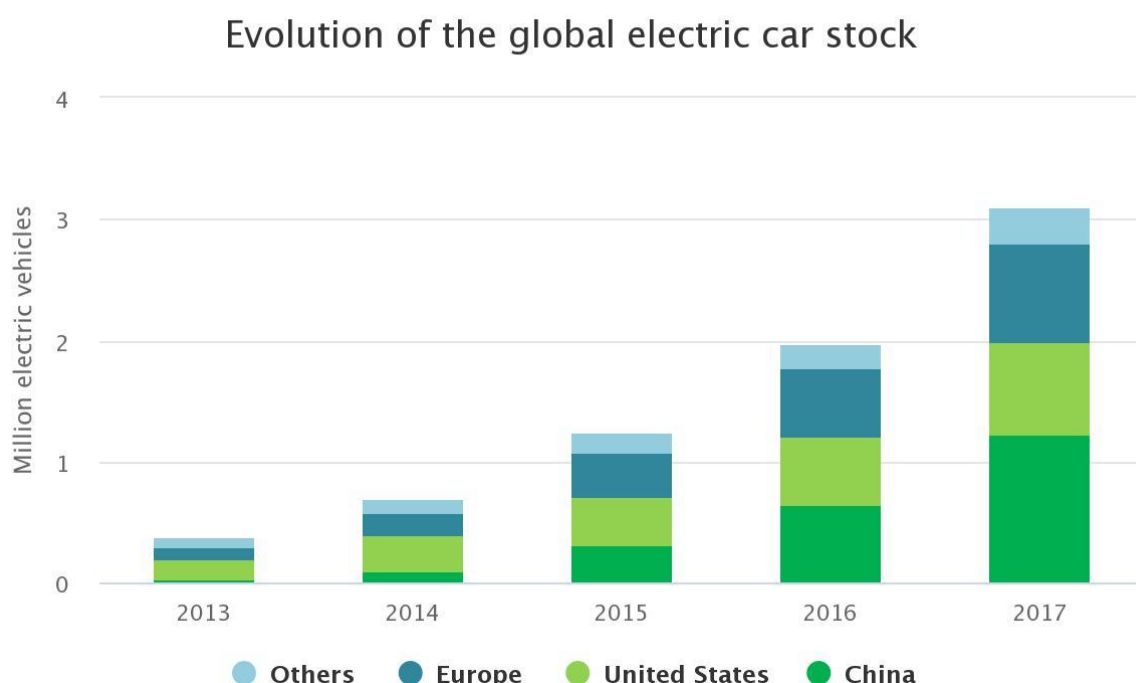
1. Wstęp	9
2. Cel i zakres pracy	11
2.1. Geneza problemu	11
2.2. Cel pracy	11
2.3. Związek z innymi pracami w rozpatrywanym obszarze	11
2.4. Założenia i zakres pracy	11
2.5. Układ pracy	12
3. Sformułowanie zadania inżynierskiego	13
3.1. Przeznaczenie systemu	13
3.2. Analiza wymagań funkcjonalnych	13
3.2.1. Wprowadzanie nowych elementów do systemu	14
3.2.2. Interakcja z użytkownikami	16
3.2.3. Monitorowanie pracy systemu	17
3.2.4. Interakcja z rynkiem energii	17
3.3. Analiza wymagań нефункциональных	18
4. Zastosowane narzędzia	19
4.1. Wirtualizacja Projektu	19
4.2. Warstwa aplikacyjna	19
4.3. Protokół komunikacyjny	20
4.4. Baza danych	20
4.5. Integracja z agregatorem	20
5. Opis rozwiązań	21
5.1. Model bazy danych	21
5.2. Panel administracyjny	23
5.3. Panel agregatora	23
5.4. Interfejs	24
5.4.1. Zarządzanie tokenami	24
5.4.2. Składanie planów ładowania	24
5.4.3. Udostępnianie decyzji	26
5.5. Moduł agregujący	27
5.5.1. Przygotowanie danych do procesu agregacji	28
5.5.2. Odebranie wyników procesu agregacji	29

5.5.3. Uruchomienie kolejnych etapów procesów sprzedaży	29
5.5.4. Odebranie wyników z procesu dezagregacji	29
5.6. Konteneryzacja	31
6. Praca z systemem	32
6.1. Wymagania	32
6.2. Pobranie projektu	32
6.3. Generowanie przykładowych danych.....	32
6.4. Uruchomienie projektu	33
6.5. Utworzenie konta użytkownika	33
6.6. Wprowadzenie węzła, punktu ładowania, samochodu elektrycznego .	34
6.7. Złożenie planu ładowania.....	35
6.8. Uruchomienie procesu agregacji.....	38
6.9. Odebranie planu ładowania	40
7. Testy.....	41
7.1. Testy automatyczne aplikacji	41
7.1.1. Wykorzystane narzędzia	41
7.1.2. Uruchomienie	41
7.1.3. Zakres	41
7.2. Testy wydajnościowe	41
8. Zakończenie	43
9. Literatura	44
10. Spis rysunków	45
11. Spis tabel.....	46

1. Wstęp

Według danych opublikowanych przez International Energy Agency liczba samochodów elektrycznych na świecie, jak zostało to przedstawione na rysunku 1.1 wzrasta z roku na rok.

Rysunek 1.1 Liczba samochodów elektrycznych w latach 2013-2017



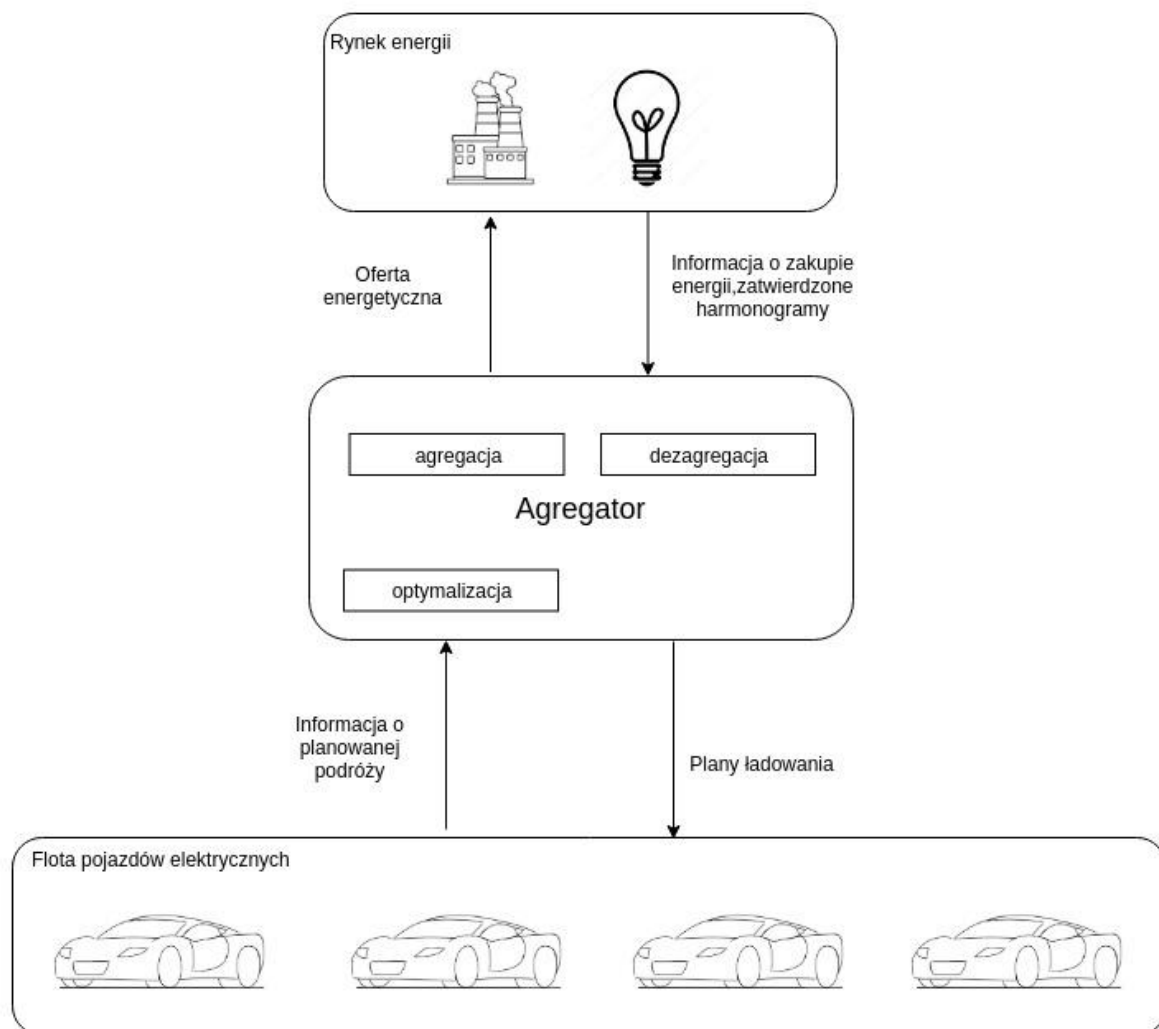
Źródło: <https://www.iea.org/tcep/transport/evs/>

Wzrastająca liczba samochodów elektrycznych niesie za sobą szereg korzyści, ale również wyzwań. Jednym z kluczowych wyzwań stawianych przed infrastrukturą, jest ładowanie pojazdów elektrycznych. Oczywiście możemy pozwolić użytkownikom na ładowanie samochodów o dowolnych porach, lecz może to powodować destabilizację rynku energetycznego. Ekspersi są zgodni, że podejściem właściwym jest inteligentne ładowanie, na przykład w [1] pokazano, że dzięki zastosowaniu modelu inteligentnego ładowania, użytkownik końcowy może zaoszczędzić od 5% do ponad 50% w stosunku do ładowania niezaplanowanego.

W celu zarządzania flotą rozproszonych pojazdów elektrycznych, ich lokalizacjami i planami ładowania, zaproponowano nową encję elektryczną: agregator pojazdów elektrycznych. Agregator jest operatorem w sieci reprezentującym grupę pojazdów elektrycznych na rynku elektroenergetycznym. Dlaczego pojazd elektryczny miałby korzystać z usług agregatora? Agregator reprezentujący grupę pojazdów elektrycznych może negocjować ceny na rynku elektroenergetycznym przy optymalizowaniu planów ładowania pojazdów. Dlaczego pojazd nie mógłby sam działać na rynku elektroenergetycznym? Zapotrzebowanie na energię poszczególnych pojazdów elektrycznych są za małe, aby móc je sprzedawać.

Zadania agregatora mogą być różne, w zależności od roli, jaką przyjmie on na rynku oraz dostępnych na tym rynku mechanizmów wspierających obrót energią z udziałem pojazdów elektrycznych. Ta praca rozważa sytuację, w której agregator jest jednostką odpowiedzialną za planowanie i zakup energii elektrycznej dla samochodów elektrycznych na dzień wcześniej. Agregator optymalizuje ofertę zakupu energii elektrycznej na podstawie planów ładowania, jakie udostępniają mu pojazdy elektryczne. Proces ten został schematycznie przedstawiony na rysunku 1.2. Udział kierowców pojazdów elektrycznych w systemie jest dobrowolny i podyktowany korzyścią w postaci niższych cen energii.

Rysunek 1.2 Schemat scentralizowanego zarządzania flotą pojazdów elektrycznych



Źródło: własne

2. Cel i zakres pracy

2.1. Geneza problemu

Motywacją do zrealizowania przeze mnie aplikacji, jest brak narzędzi obsługujących siećową komunikację z samochodami elektrycznymi wykorzystującą specyficzne dane. Dane te to plany ładowania na kolejny dzień zawierające punkty ładowania i godziny obecności w tych punktach. W związku z czym podmioty już działające na rynku nie mają możliwości efektywnego planowania ładowania i minimalizacji kosztów zakupu energii dla swojej floty.

2.2. Cel pracy

Celem pracy dyplomowej jest stworzenie aplikacji internetowej do wspomagania komunikacji w systemie zarządzania inteligentnego ładowania floty pojazdów elektrycznych. Aplikacja powinna umożliwić składanie planów ładowania przez samochody elektryczne, aby następnie przygotować dane w odpowiednim formacie dla procesu agregacji. Agregacja, symulowanie odpowiedzi rynku elektroenergetycznego i dezagregacja są wykonywane przez zewnętrzne narzędzie. Następnym krokiem jest odebranie zdezagregowanych planów ładowania i udostępnienie ich użytkownikom samochodów elektrycznych. Aplikacja powinna umożliwić zarządzanie i monitorowanie systemu.

2.3. Związek z innymi pracami w rozpatrywanym obszarze

Założeniem pracy jest możliwość prostej integracji narzędzia z możliwie szerokim spektrum innych funkcjonalności. W szczególności, narzędzie to ma możliwość zastosowania opracowanych wcześniej algorytmów agregacji i dezagregacji. W pracy wykorzystano implementację agregacji i dezagregacji pracy inżynierskiej Karoliny Drabarz [2].

2.4. Założenia i zakres pracy

W ramach tej pracy skupiono się na stworzeniu aplikacji internetowej, modelowaniu bazy danych, tworzeniu interfejsów sieciowych, tworzeniu i skonfigurowaniu środowiska w którym działa aplikacja.

Warto podkreślić, że cały proces pobierania danych i generowania planów ładowania odbywa się z jednodniowym wyprzedzeniem w stosunku do rzeczywistego ładowania pojazdów i że proces ten jest powtarzany każdego dnia.

Praca zakłada synchroniczność odpowiedzi rynku energii, oznacza to, że administrator uruchamiający agregację otrzymuje odpowiedź od rynku elektroenergetycznego od razu, co jest pewnym uproszczeniem w stosunku do realnego procesu zarządzania ładowaniem pojazdów elektrycznych, w którym na odpowiedź należy poczekać nawet kilka godzin.

Na potrzeby pracy przyjęto również uproszczony model zarządzania parametrami agregacji zawierającym tylko niezbędne parametry. W rzeczywistym procesie zbiór parametrów byłby większy.

2.5. Układ pracy

W rozdziale 3. *Sformułowanie zadania inżynierskiego* znajdziemy wymagania funkcjonalne i нефункционаłne projektu. Rozdział 4. *Zastosowane narzędzia* opisuje technologie, biblioteki i frameworki wykorzystane w projekcie. W rozdziale 5. *Opis rozwiązań* opisują implementację poszczególnych problemów wraz z wskazaniem plików źródłowych. Rozdział 6. *Praca z systemem* przedstawia sposób używania konkretnych funkcji systemu. W zakończeniu, w rozdziale 7. *Testy* opisują, w jaki sposób aplikacja jest testowana.

3. Sformułowanie zadania inżynierskiego

3.1. Przeznaczenie systemu

Aplikacja jest narzędziem, z którego korzystają użytkownicy samochodów elektrycznych jako klienci aplikacji. Użytkownicy samochodów elektrycznych składają swoje plany ładowania na kolejny dzień, które aplikacja zapisuje. Aplikacja przygotowuje odpowiednio plany ładowania do procesu agregacji. Agregator przeprowadza proces agregacji i kupuje energię elektryczną, a następnie przekazuje plany zakupionej energii aplikacji, która zapisuje je i udostępnia je jako plany ładowania samochodom elektrycznym.

Aplikacja udostępnia administratorowi możliwość przeprowadzania procesu agregacji i dezagregacji z przekazaniem parametrów procesu i wyświetla rezultat procesu agregacji i dezagregacji.

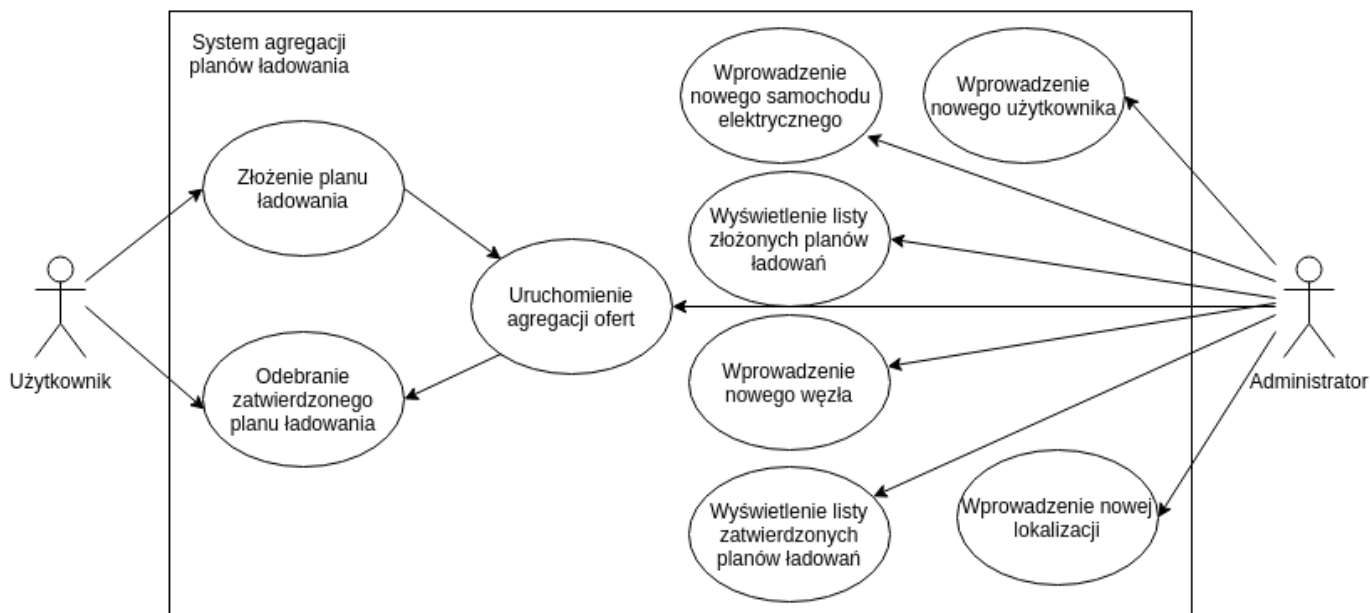
3.2. Analiza wymagań funkcjonalnych

Wymagania funkcjonalne to konkretne aspekty funkcjonowania aplikacji, które definiują działania, przeznaczone dla użytkowników oprogramowania. Ich specyfikacja jest kluczowym punktem planowania projektu, ponieważ ich odpowiednie zdefiniowanie może znacznie uprościć implementację [3].

Najpopularniejszym sposobem określenia wymagań funkcjonalnych są przypadki użycia (ang. use cases). Jest to model polegający na definiowaniu zadań, jakie użytkownicy systemu (ang. actors) chcą za pomocą tego systemu wykonać. Każde z zadań można opisać jako listę kroków, w których użytkownik wchodzi w interakcję z systemem, by na końcu odebrać potrzebny mu wynik.

Graficzną reprezentacją modelu jest diagram przypadków użycia, który poprzez ikony aktorów i owale reprezentujące przypadki użycia pokazuje, jakie relacje zachodzą między nimi [4].

Rysunek 1.3 Diagram przypadków użycia



Źródło: własne

W niniejszym rozdziale zostaną przedstawione przypadki użycia aplikacji. Do opisu pojedynczego przypadku:

1. Nazwa - czytelna dla człowieka
2. Identyfikator - Kod alfanumeryczny jednoznacznie identyfikujący przypadek użycia
3. Aktor - Kategoria użytkowników systemu
4. Scenariusz główny - Tekstowy zapis interakcji z systemem
5. Scenariusz alternatywny - Tekstowy zapis sytuacji nadzwyczajnych uniemożliwiające dorowadzenie procedury do końca.

3.2.1. Wprowadzanie nowych elementów do systemu

Nazwa i identyfikator: Utworzenie konta użytkownika A01
Aktorzy: Administrator
<p>Scenariusz główny:</p> <ol style="list-style-type: none"> 1. Administrator loguje się do panelu administracyjnego 2. Administrator wybiera opcję dodania nowego użytkownika 3. Administrator wprowadza nazwę użytkownika i hasło 4. System zapisuje nowego użytkownika 5. Administrator przekazuje nazwę użytkownika i hasło nowemu użytkownikowi
<p>Scenariusz alternatywny 1: Nazwa użytkownika jest już zajęta</p> <ol style="list-style-type: none"> 1. Kroki od 1 do 3 Jak w scenariuszu głównym

2. Administrator wybiera inną nazwę użytkownika

Nazwa i identyfikator: Wprowadzenie nowego węzła A02

Aktorzy: Administrator

Scenariusz główny:

1. Administrator loguje się do panelu administracyjnego
2. Administrator wybiera opcję dodania nowego węzła
3. Administrator wprowadza nazwę węzła
4. System zapisuje nowy węzeł

Nazwa i identyfikator: Wprowadzenie nowej lokalizacji A03

Aktorzy: Administrator

Scenariusz główny:

1. Administrator loguje się do panelu administracyjnego
2. Administrator wybiera opcję dodania nowej lokalizacji
3. Administrator wprowadza adres lokalizacji
4. System zapisuje nową lokalizację

Nazwa i identyfikator: Wprowadzenie nowego samochodu elektrycznego A04

Aktorzy: Administrator

Scenariusz główny:

1. Administrator loguje się do panelu administracyjnego
2. Administrator wybiera opcję dodania nowego samochodu
3. Administrator wprowadza:
 - a. Minimalną pojemność baterii
 - b. Maksymalną pojemność baterii
 - c. Maksymalną moc ładowania
4. Administrator wybiera właściciela samochodu spośród zarejestrowanych użytkowników
5. System zapisuje nowy samochód elektryczny

3.2.2. Interakcja z użytkownikami

Nazwa i identyfikator: Złożenie planu ładowania U01
Aktorzy: Użytkownik
Scenariusz główny: <ol style="list-style-type: none">1. Użytkownik wysyła plan ładowania na kolejny dzień (miejsca, dokładne czasy ładowania, identyfikator samochodu elektrycznego)2. System uwierzytelnia użytkownika3. System autoryzuje użytkownika4. System sprawdza poprawność złożonego planu ładowania5. System zapisuje plan ładowania i odsyła użytkownikowi potwierdzenie
Scenariusz alternatywny 1: Użytkownik nie jest uwierzytelniony <ol style="list-style-type: none">1. Kroki 1 Jak w scenariuszu głównym2. System zwraca użytkownikowi informację że nie jest on uwierzytelniony
Scenariusz alternatywny 2: Użytkownik nie jest właścicielem samochodu <ol style="list-style-type: none">1. Kroki od 1 do 2 Jak w scenariuszu głównym2. System zwraca użytkownikowi informację że nie jest on właścicielem samochodu
Scenariusz alternatywny 3: Plan ładowania jest niepoprawny <ol style="list-style-type: none">1. Kroki od 1 do 3 Jak w scenariuszu głównym2. System zwraca użytkownikowi informację gdzie z złożonym planie ładowania pojawiły się błędy

Nazwa i identyfikator: Odebrania planu ładowania U02
Aktorzy: Użytkownik
Scenariusz główny: <ol style="list-style-type: none">1. Użytkownik wysyła prośbę o odebranie plan ładowania na kolejny dzień dla konkretnego samochodu elektrycznego2. System uwierzytelnia użytkownika3. System autoryzuje użytkownika4. System wysyła zatwierdzony plan ładowania użytkownikowi
Scenariusz alternatywny 1: Użytkownik nie jest uwierzytelniony <ol style="list-style-type: none">1. Kroki 1 Jak w scenariuszu głównym2. System zwraca użytkownikowi informację że nie jest on uwierzytelniony

Scenariusz alternatywny 2: Użytkownik nie jest właścicielem samochodu 1. Kroki od 1 do 2 jak w scenariuszu głównym 2. System zwraca użytkownikowi informację że nie jest on właścicielem samochodu
--

Scenariusz alternatywny 3: Brak zatwierdzonego planu ładowania 1. Kroki od 1 do 3 jak w scenariuszu głównym 2. System zwraca użytkownikowi informację o braku zatwierdzonego planu ładowania
--

3.2.3. Monitorowanie pracy systemu

Nazwa i identyfikator: Wyświetlenie listy złożonych planów ładowania A05
--

Aktorzy: Administrator

Scenariusz główny: 1. Administrator loguje się do panelu administracyjnego 2. Administrator wybiera opcję wyświetlenia listy złożonych planów ładowania 3. System wyświetla listę złożonych planów ładowania

Nazwa i identyfikator: Wyświetlenie listy decyzji planów ładowania A06
--

Aktorzy: Administrator

Scenariusz główny: 1. Administrator loguje się do panelu administracyjnego 2. Administrator wybiera opcję wyświetlenia listy decyzji planów ładowania 3. System wyświetla listę zatwierdzonych planów ładowania
--

3.2.4. Interakcja z rynkiem energii

Nazwa i identyfikator: Uruchomienie agregacji ofert A07

Aktorzy: Administrator

Scenariusz główny: 1. Administrator wybiera opcję uruchomienia agregacji ofert 2. Administrator wybiera metodę agregacji, pokrycie zapotrzebowanie energii, minimalną wartość oferty 3. System grupuje plany ładowania względem wybranej metody i przygotowuje

- oferty w odpowiednim formacie i uruchamia proces agregacji
4. System zapisuje wynik procesu agregacji
 5. System wystawia ofertę na rynek elektroenergetyczny
 6. System odbiera plan ładowania
 7. System uruchamia proces dezagregacji
 8. System wyświetla informacje o przeprowadzonym procesie agregacji i dezagregacji

Scenariusz alternatywny 1: Proces agregacji nie powiódł się

1. Kroki od 1 do 4 jak w scenariuszu głównym
2. System odbiera informację od agregatora o nieudanym procesie agregacji
3. System wyświetla informację o niepoprawnej agregacji

Scenariusz alternatywny 2: Proces dezagregacji nie powiódł się

1. Kroki od 1 do 6 jak w scenariuszu głównym
2. System zapisuje informację o niepowodzeniu dezagregacji

3.3. Analiza wymagań niefunkcjonalnych

W rozdziale zostaną przedstawione wymagania niefunkcjonalne, czyli takie, które wskazują ograniczenia, przy których zachowaniu system powinien realizować swoje funkcje.

1. Przewidywana liczba samochodów elektrycznych, którą aplikacja powinna obsługiwać to od 1000 do 10 000. Odebranie planów ładowania jest dostępne dla samochodów od pewnej godziny co jest związane z ustaleniem planów ładowania w jednej chwili. Aplikacja powinna obsługiwać 10 000 żądań w 10 minut, czyli około 17 żądań na sekundę.
2. System powinien zapewnić bezpieczeństwo danych, dane użytkowników i ich planów ładowania nie powinny być dostępne publicznie.
3. System w razie awarii powinien w czasie nie dłuższym niż 4 godziny wrócić do działania.
4. System powinien używać otwartego protokołu komunikacji z klientami tj. samochodami elektrycznymi.
5. System powinien być otwarty na implementację algorytmów agregacji i dezagregacji.
6. System powinien być przenośny.
7. System powinien być rozszerzalny.

4. Zastosowane narzędzia

Aplikacja wykorzystuje następujące technologie:

- Python3.7
- Django v2.1.3
- Django REST framework v3.9.2
- marshmallow v3.0.0
- Pytest
- Docker v18.09.5
- PostgreSQL v9.6
- JSON

4.1. Wirtualizacja Projektu

Docker jest technologią, która pozwala umieścić aplikację i wszystkie jej zależności w wirtualnym kontenerze, który z punktu widzenia aplikacji jest odrębną instancją środowiska uruchomieniowego. Każdy kontener posiada wydzielony obszar na dysku, na którym znajduje się zainstalowany obraz systemu operacyjnego i wszystkich bibliotek potrzebnych do działania aplikacji.

Dzięki skonteneryzowaniu aplikacji bardzo łatwo uruchomić ją w środowisku deweloperskim co jest niezwykle przydatne, przy pracy nad projektem gdzie wielokrotnie uruchamiamy i zamykamy aplikację. Konteneryzacja jest też przydatna przy uruchomieniu testów w odseparowanym środowisku, po którego zamknięciu nie pozostają żadne ślady w macierzystym systemie operacyjnym.

Kolejną zaletą jest przenośność aplikacji między środowiskami systemu Linux, co otwiera aplikacji możliwość bardzo łatwego przeniesienia do chmury.

4.2. Warstwa aplikacyjna

Do stworzenia aplikacji internetowej zdecydowano się na użycie framework'a Django. Jest to framework z otwartymi źródłami przeznaczony do tworzenia aplikacji internetowych oparty na języku Python. Główną motywacją wyboru było zapewnienie pełnego zestawu potrzebnych mi narzędzi zarówno ORM¹u, jak i bibliotek HTTP. Motywacją była też szybkość i łatwość tworzenia oprogramowania.

Django zapewnia mechanikę warstwy aplikacji, czyli pozwala na obsługiwanie żądań sieciowy, w tym:

- udostępnianie danych
- walidację danych
- autoryzację
- uwierzytelnianie

¹ Odzworowanie architektury obiektowej na relacyjne struktury bazy danych
https://pl.wikipedia.org/wiki/Mapowanie_obiektowo-relacyjne

Poprzez ORM pozwala w łatwy sposób modyfikować dane znajdujące się w bazie danych.

4.3. Protokół komunikacyjny

Jako protokół komunikacyjny między aplikacją a użytkownikami wybrano tekstowy format wymiany danych JSON, który został opisany w dokumencie [5]. Format jest niezależny od języka programowania, co pozwala na implementację klientów w dowolnej technologii.

4.4. Baza danych

Jako bazę danych wybrano PostgreSQL, która jest relacyjną bazą danych z otwartymi źródłami.

4.5. Integracja z agregatorem

Integracja polega na odpowiednim przygotowaniu danych, które zrozumie agregator, a następnie, po przeprowadzonym procesie agregacji, odebraniu danych, które agregator wyprodukuje. Formaty danych na wejście i wyjście agregatora są zupełnie inne niż znormalizowane dane przechowywane w bazie danych. W celu przeprowadzenia konwersji danych pomiędzy bazą danych a agregatorem, użyto biblioteki marshmallow, która oferuje konwersję złożonych typów danych z i do obiektów języka Python.

5. Opis rozwiązań

Rozdział będzie poświęcony opisowi pracy własnej w implementacji projektu wraz ze wskazaniem gdzie w projekcie możemy znaleźć kod źródłowy implementujący dany problem. Projekt dzieli się na następujące moduły:

- *thesis/scripts* - skrypty uruchomieniowe
- */thesis/thesis/aggregator* - ustawiania projektu
- */thesis/thesis/aggregator_simulation/* - symulacje agregatora
- */thesis/thesis/scripts/generate_demo.py* - moduł generując przykładowe dane
- */thesis/thesis/apps/aggregator_integration* - moduł odpowiedzialny za integrację z agregatorem
- */thesis/thesis/apps/schedules* - moduł odbierający plany ładowania
- */thesis/thesis/apps/decisions* - moduł udostępniający decyzję rynku energetycznego

5.1. Model bazy danych

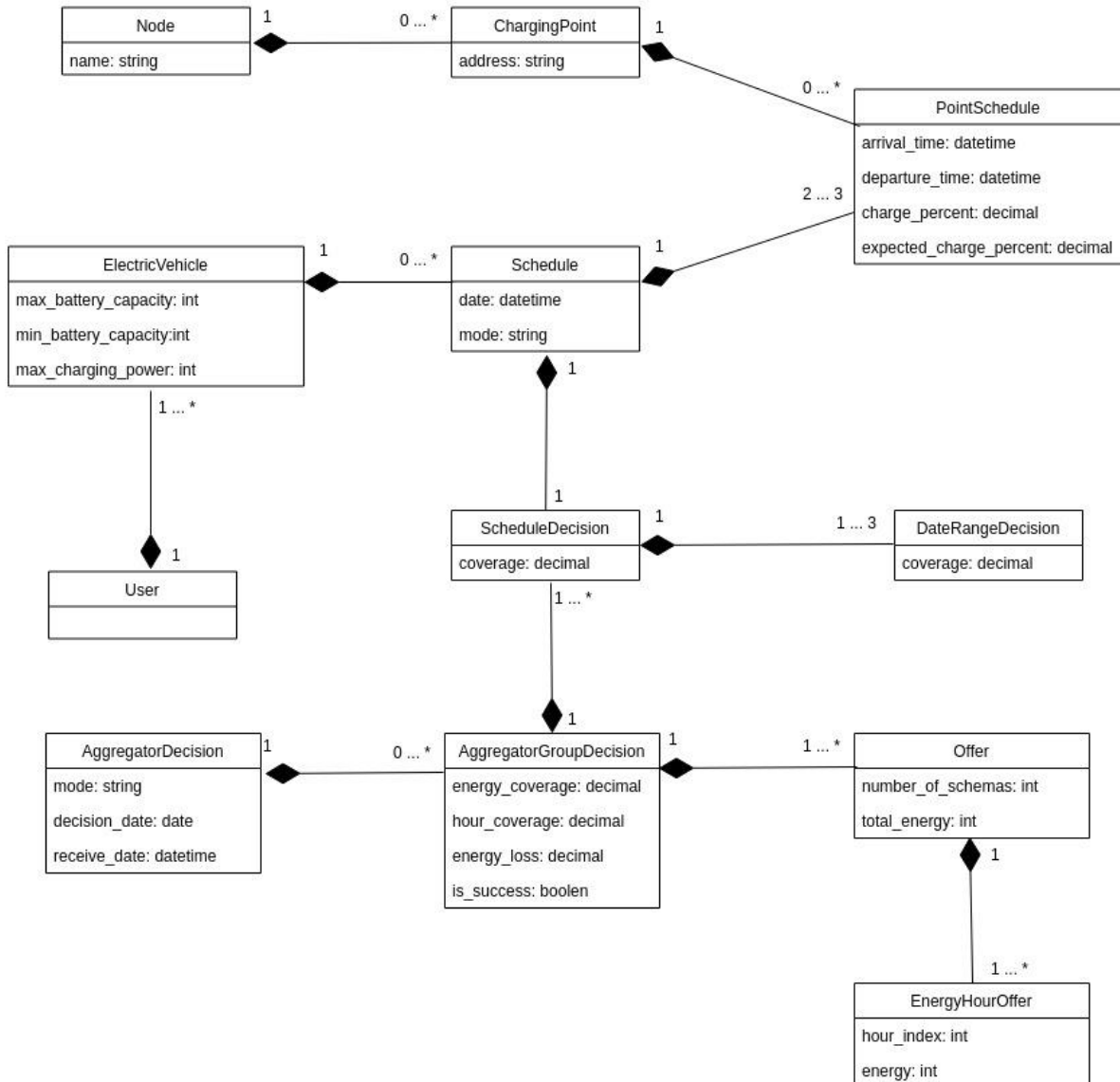
Aplikacja zawiera 12 encji:

1. User - klient systemu, właściciel jednego lub więcej pojazdu elektrycznego.
2. ElectricVehicle - samochód elektryczny,
3. Node - węzeł sieci elektrycznej, do którego należą punkty ładowania.
4. ChargingPoint - punkt ładowania pojazdu elektrycznego.
5. Schedule - Plan ładowania składany przez pojazd elektryczny. Plan zawiera w sobie z 2 lub 3 miejsca ładowania(PointSchedule) i informację o aktualnym stanie baterii i energii potrzebne do przebycia zaplanowanej drogi.
6. PointSchedule - Miejsce ładowanie, które samochód elektryczny deklaruje w swoim planie ładowania, zawiera informację o godzinach, w jakich samochód planuje znajdować się w konkretnym punkcie ładowania.
7. AggregatorDecision - Decyzja dla planów ładowania dla konkretnego dnia, składa się z wyników agregacji i dezagregacji dla konkretnych grup.
8. AggregatorGroupDecision - Informacje o dezagregacji dla konkretnej grupy samochodów elektrycznych. Zawiera plany ładowania (ScheduleDecision) dla poszczególne grupy wybranej w procesie agregacji.
9. Offer - Informacja o procesie agregacji dla grupy samochodów elektrycznych. Zawiera informację o liczbie planów ładowania w tej grupie i żądanej energii.
10. EnergyHourOffer - Żądana energia dla konkretnych godzin w grupie.
11. ScheduleDecision - Plan ładowania po zakupie energii elektrycznej dla deklaracji planu ładowania (PointSchedule). Zawiera godziny (DateRangeDecision) w jakich samochód elektryczny powinien ładować się w zadeklarowanym miejscu i pokrycie potrzeb energii, jakie decyzja spełnia wobec deklaracji.

12. DateRangeDecision - Zakresy czasu ładowania, jakie zostały zdecydowane w procesie kupna energii

Na schemacie 5.1 znajduje się diagram w języku UML przedstawiający encję wraz z relacjami, jakie zachodzą między nimi.

Schemat 5.1 Schemat UML relacji encji



Źródło: własne

Implementację wyżej wymienionych encji można znaleźć:

- `/thesis/thesis/apps/decisions/models.py`
- `/thesis/thesis/apps/schedules/models.py`

5.2. Panel administracyjny

Framework django oferuje panel administracyjny, dzięki któremu możemy przeglądać dane znajdujące się w bazie danych, to co musimy zrobić, to skonfigurować jak mają wyglądać poszczególne widoki. Konfigurację możemy znaleźć:

- `/thesis/thesis/apps/decisions/admin.py`
- `/thesis/thesis/apps/schedules/admin.py`

5.3. Panel agregatora

Panel do uruchomienia procesu agregacji znajduje się pod adresem `/auction/` umieszczono tam informację o ilości czasu, przez jaki jeszcze samochody elektryczne mogą składać plany ładowania oraz przycisk odpowiedzialny za uruchomienie procesu agregacji i dezagregacji. Po naciśnięciu przycisku zostaniemy przeniesieni do formularza, w którym możemy skonfigurować proces agregacji danymi:

- *method* - Dostępne są 2 metody agregacji: wszystkie plany ładowania na raz i z podziałem na lokalizację
- *coverage* - Stopień pokrycia zapotrzebowania zgłaszanego przez pojazdy elektryczne
- *Minimum energy offer* - minimalna łączna suma energii w ofercie

Po zatwierdzeniu formularza proces agregacji i dezagregacji zostanie uruchomiony. Po przeprowadzonych procesach zostaną wyświetlone dane dla każdej zagregowanej grupy takie jak:

- Pokrycie zapotrzebowania energii
- Całkowita energia żądana
- Zapotrzebowanie energetyczne dla konkretnych godzin

Kod odpowiedzialny za wygląd panelu znajdziemy:

- `/thesis/thesis/apps/schedules/templates/`

Kod odpowiedzialny za logikę wyświetlania stron w klasie *TriggerAggregationView* w pliku:

- `/thesis/thesis/apps/schedules/views.py`

Kod odpowiedzialny za przeprowadzenie procesu agregacji i dezagregacji w klasie *AggregationService* w pliku:

- `/thesis/thesis/apps/aggregator_integration/services.py`

5.4. Interfejs

Do stworzenia interfejsu zgodnego z stylem architektury oprogramowania REST² posłużyłem się biblioteką *django REST framework* (<https://www.django-rest-framework.org/>), która jest narzędziem zintegrowanym z *Django* do tworzenia sieciowych interfejsów.

Aplikacja jako interfejs dla użytkownika udostępnia następujące endpointy:

- */api-token-auth/* - interfejs do zarządzania tokenami tokenów
- */charging-schedules/* - interfejs do składania planów ładowania przez użytkowników
- */charging-schedules-decisions/* - interfejs do udostępniania decyzji dla planów ładowania

5.4.1. Zarządzanie tokenami

Każde żądanie do interfejsu składania czy odbierania ofert przez użytkowników samochodów elektrycznych musi być uwierzytelnione. Odbywa się to poprzez dodanie nagłówka *Authorization* z tokenem.

W celu uzyskania tokena musimy skorzystać z interfejsu */api-token-auth/* wymaga on:

- nazwy użytkownika
- hasła

Przykład żądania:

```
curl --data "username=nazwa" --data "password=haslo" -X POST 127.0.0.1:8000/api-token-auth/
```

Przykład odpowiedzi:

```
{"token": "c0a7456526025babd6cbc83fe39f7e29ee644ead"}
```

Następnie do każdego żądania musimy dodać nagłówek:

```
"Authorization: Token c0a7456526025babd6cbc83fe39f7e29ee644ead"
```

5.4.2. Składanie planów ładowania

Plan ładowania, jaki może złożyć uwierzytelniony użytkownik dla jednego ze swoich samochodów elektrycznych, może być dwóch typów:

1. *Home-home* - gdy deklaruje, że następnego dnia będzie ładował się tylko w jednej lokalizacji w 2 blokach czasowych
2. *Home-work-home* - gdy deklaruje, że następnego dnia będzie ładował się w dwóch lokalizacjach w 3 blokach czasowych

Bloki czasowe, jakie definiuje użytkownik, nie mogą być puste i nie mogą nachodzić na siebie nawzajem.

² Styl architektury dla serwisów internetowych
https://en.wikipedia.org/wiki/Representational_state_transfer

Interfejs udostępnia metodę *POST*, która wymaga pól:

- `electric_vehicle` - identyfikator samochodu elektrycznego
- `mode` - typ planowania: *hh* lub *hwh*
- `charge_percent` - aktualny stan naładowania samochodu
- `trip_percent` - ilość energii potrzeba na przebycie zaplanowanej trasy
- `point_schedules` - lista miejsc ładowania gdzie struktura elementu zawiera:
 - `point` - identyfikator punktu ładowania
 - `arrival_time` - czas przyjazdu do punktu ładowania
 - `departure_time` - czas wyjazdu z punktu ładowania

```
{
  "mode": "hh",
  "point_schedules": [
    {
      "departure_time": "2019-03-10T08:00:00Z",
      "arrival_time": "2019-03-10T15:30:00Z",
      "point": 4
    },
    {
      "departure_time": "2019-03-10T17:30:00Z",
      "arrival_time": "2019-03-10T23:30:00Z",
      "point": 4
    }
  ],
  "electric_vehicle": 5,
  "trip_percent": 60,
  "charge_percent": 10
}
```

Odpowiedzi zwracane przez interfejs zostały przedstawione w tabeli 5.1

Tabela 5.1 Tabela odpowiedzi, interfejs do składania planów ładowania

Kod odpowiedz	Opis
201	Przyjęto plan ładowania
401	Żądanie nieautoryzowane. Brak tokenu.
403	Próba złożenia planu do samochodu do którego nie ma uprawnień
400	Niepoprawne dane możliwe przyczyny: 1. Samochód o podanym identyfikatorze nie istnieje 2. Niepoprawna liczba miejsc ładowania

Źródło: własne

Do walidacji danych wejściowych posłużyłem się serializatorami³ które możemy znaleźć:

- `/thesis/thesis/apps/schedules/serializers.py`

w klasach: `ScheduleSerializer` i `PointScheduleSerializer`

Kod odpowiedzialny za uwierzytelnienie użytkownika znajdziemy w:

- `/thesis/thesis/apps/schedules/views.py`

w klasie `ScheduleViewSet`

Testy pokrycia przypadków użycia znajdziemy w:

- `/thesis/thesis/apps/schedules/tests/test_views.py`

5.4.3. Udostępnianie decyzji

Po uruchomieniu przez administratora procesu agregacji i sprzedaniu zagregowanych ofert na rynku energetycznym, użytkownik może odebrać decyzję o planie ładowania dla kolejnego dnia, jak i historyczne dane o decyzjach z poprzednich dni.

Endpoint udostępnia metodę GET, która udostępnia dane jako listę decyzji, z których każda zawiera:

- pokrycie żądanej energii
- listę przedziałów czasowych ładowania

W żądaniu w parametrach url należy podać identyfikator samochodu elektrycznego, dla którego wykonujemy żądanie.

Przykład żądania o plan ładowania na kolejny dzień:

```
curl -X GET 127.0.0.1:8000/charging-schedules-decisions/next_day/?electric_vehicle=34
```

Przykład żądania o plan ładowania na dowolny dzień w historii:

```
curl -X GET 127.0.0.1:8000/charging-schedules-decisions/?electric_vehicle=34&date=2019-07-24
```

³ Więcej o warstwie serializacji: <https://www.django-rest-framework.org/api-guide/serializers/>

Przykład odpowiedzi:

```
{
  "coverage": "30.5",
  "schedule_decision_date_ranges": [
    {
      "start_time": "07:30:00",
      "end_time": "13:00:00"
    },
    {
      "start_time": "15:00:00",
      "end_time": "20:00:00"
    }
  ]
}
```

Odpowiedzi zwracane przez interfejs zostały przedstawione w tabeli 5.1

Tabela 5.2 Tabela odpowiedzi, interfejs do pobierania planów ładowania

Kod odpowiedzi	Opis
200	Zwracane są decyzje o planie ładowania.
403	Próba pobrania planu dla samochodu dla którego nie ma uprawnień.
404	Brak decyzji rynku energetycznego dla kolejnego dnia.
404	Brak decyzji dla konkretnego samochodu.

Źródło: własne

Kod odpowiedzialny za uwierzytelnienie użytkownika i wyszukanie odpowiednich ofert znajdziemy w:

- `/thesis/thesis/apps/decisions/views.py`
w klasie `ChargingSchedulesDecisionViewSet`

Testy pokrycia przypadków użycia znajdziemy w:

- `/thesis/thesis/apps/decisions/tests/test_api.py`

5.5. Moduł agregujący

W celu agregacji ofert, złożenia ofert na rynku energii, a następnie odebraniu wyników desegregacji posłużyłem się projektem opisanym w rozdziale 2.3. Związek z innymi pracami w rozpatrywanym obszarze nazywając go w dalszej części agregatorem. W niniejszym rozdziale opiszę integrację i użycie agregatora.

Częścią autorską modułu jest:

1. Przygotowanie danych dla agregatora

2. Uruchomienie kolejnych etapów aukcji:
 1. Agregacji ofert
 2. Złożenie ofert na rynku
 3. Odebranie wyników aukcji od agregatora
3. Odebranie danych od agregatora

Głównym problemem i wyzwaniem integracji z agregatorem jest format danych, jaki przyjmuje i zwraca agregator, który jest zupełnie inny niż znormalizowany format, w jakim dane są trzymane w bazie danych. Agregator posługuje się formatem JSON. Do rozwiązania problemu użyłem biblioteki *marshmallow* <https://marshmallow.readthedocs.io/en/3.0/>, która umożliwia konwersję złożonych typów danych do i z formatu JSON.

Kod odpowiedzialny za przeprowadzenie całego procesu, który wykorzystuje moduły opisane w kolejnych rozdziałach, znajdziemy w:

- `/thesis/thesis/apps/aggregator_integration/services.py`

5.5.1. Przygotowanie danych do procesu agregacji

Aby uruchomić proces agregacji należy przekazać agregatorowi listę planów ładowania w formacie:

```
{
  "tripsData": [
    {
      'plugInSchedule': [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, ..., 0, 0, 0, 0],
      'battery': {
        'batteryProfile': {
          'hourPercentageCharge': 10.0,
          'capacity': 100.0
        },
        'inputEnergyLevel': 20.0,
        'outputEnergyLevel': 70.0
      },
      "id": 132
    },
    ...
  ]
}
```

tripsData to lista planów ładowania gdzie:

- *plugInSchedule* - 24 elementowa lista określająca, w których godzinach chce się ładować samochód
- *hourPercentageCharge* - o ile procent samochód może załadować się w godzinę
- *capacity* - pojemność baterii w kilowatogodzinach
- *inputEnergyLevel* - procent naładowania przed ładowaniem w danym punkcie

- *outputEnergyLevel* - wymagany procent naładowania po naładowaniu
- *id* - unikalny identyfikator samochodu

Modele z których pobierane są dane do przygotowania procesu agregacji to: *ElectricVehicle*, *Schedule*, *PointSchedule*

Kod odpowiedzialny za serializację ofert do formatu JSON znajdziemy:

- */thesis/thesis/apps/aggregator_integration/generator.py*

5.5.2. Odebranie wyników procesu agregacji

Proces agregacji agreguje wiele planów ładowania w oferty wystawiane następnie rynku elektroenergetycznym. Przykładowa oferta:

```
{
    "energyDemands": [450, 342, ..., 0],
    "totalEnergyDemand": 4573.0,
    "totalNumberOfSchemes": 135
}
```

energyDemands - lista 24 wartości zapotrzebowania energii na daną godzinę

totalEnergyDemand - zapotrzebowanie na energię całej oferty

totalNumberOfSchemes - Liczba planów ładowania zagregowanych w tej ofercie

Dane pobrane z tej struktury zostaną zapisane z modelach *Offer* i *EnergyHourOffer*

5.5.3. Uruchomienie kolejnych etapów procesów sprzedaży

Aby przeprowadzić cały proces sprzedaży ofert, należy kolejno:

1. Uruchomić proces agregacji
2. Wygenerować odpowiedzi rynku energii
3. Uruchomić proces dezagregacji

Kod odpowiedzialny za wywoływanie poszczególnych funkcjonalności agregatora znajdziemy:

- */thesis/thesis/apps/aggregator_integration/aggregator_commands.py*

5.5.4. Odebranie wyników z procesu dezagregacji

Rezultatem pracy agregatora są plany ładowania w formacie:

```

{
  "disaggregatedTripsData": [{
    "data": {
      'id': ev.id,
      'localization': 1,
      'plugInSchedule': {
        'schedule': [],
      },
      'battery': {
        'batteryProfile': {
          'hourPercentageCharge': 10.0,
          'capacity': 100.0
        },
        'inputEnergyLevel': 10.0,
        'outputEnergyLevel': 99.99
      }
    },
    "coverage": 70.0,
    "chargeHours": [0, 0, 0, 0, 0, ..., 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0]
  },
  ...
],
  "totalNumberOfSchemes": 2,
  "totalEnergyCoverage": 70.34,
  "totalHourCoverage": 60.23,
  "totalEnergyLoss": 2.45
}

```

disaggregatedTripsData - to lista planów ładowania gdzie:

- data - oryginalny plan ładowania nadesłany przez samochód elektryczny
- coverage - pokrycie energetyczne
- chargeHours - lista 24 elementów określających plan ładowania

dane dotyczące całego procesu sprzedaży ofert:

- totalNumberOfSchemas - liczba planów ładowania w procesie agregacji
- totalEnergyCoverage - pokrycie wszystkich planów ładowania
- totalHourCoverage - procent pokrycia godzin ładowania
- totalEnergyLoss - liczba kilowatogodzin nieprzydzielona do ładowania

Dane pobrane z etapu dezagregacji zostaną zapisane w modelach:

AggregatorDecision, *AggregatorGroupDecision*, *ScheduleDecision*,
DateRangeDecision.

Kod odpowiedzialny za deserializację wyników aukcji i zapisanie do bazy danych znajdziemy:

- */thesis/thesis/apps/aggregator_integration/loader.py*

5.6. Konteneryzacja

Dzięki technologii *Doker* aplikacja jest spakowana w kontener gotowy do pobrania i użycia. Aplikacja uruchamiana jest w kontenerze nie tylko podczas użytkowania, ale również podczas testów, które są uruchamiane w oddzielnym kontenerze. Kod odpowiedzialny za konteneryzację znajduje się w plikach:

- */thesis/Dockerfile*
- */thesis/docker-compose.yml*

Skrypty uruchomieniowe aplikacji i testów znajdziemy w plikach:

- */thesis/scripts/run-init*
- */thesis/scripts/run-tests*
- */thesis/scripts/run-recreate-db*

6. Praca z systemem

W niniejszym rozdziale przedstawiono całościowy scenariusz pracy z systemem wraz z przykładami, które ilustrują, gdzie znajdują się konkretne funkcje systemu. Praca administratora systemu polega na użytkowaniu panelu administracyjnego, aby zilustrować jego funkcję, posłużę się zrzutami ekranu. Interakcja użytkownika odbywa się natomiast poprzez zapytania sieciowe, w celu zilustrowania jak zapytania powinny wyglądać posłużono się programem cURL⁴. Równolegle z scenariuszem, podano przypadki użycia (p. u.) który dany fragment scenariusza pokrywa.

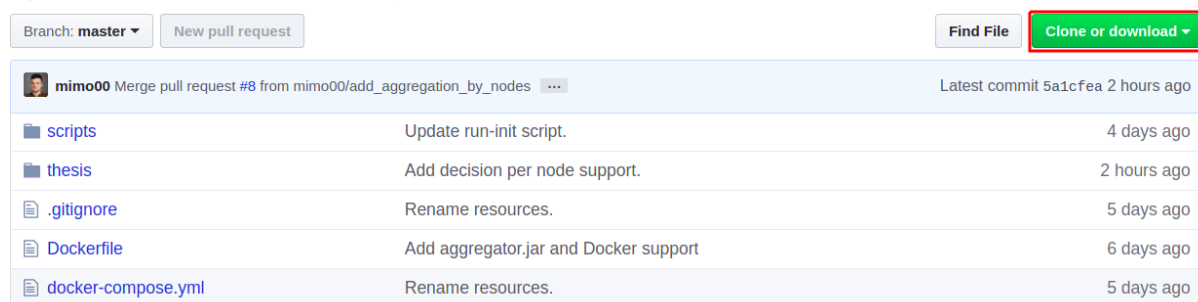
6.1. Wymagania

W celu uruchomienia projektu, należy zainstalować *Docker'a* co najmniej w wersji 18.09. W tym celu można skorzystać z instrukcji dostawcy: <https://docs.docker.com/install/linux/docker-ce/ubuntu/>

6.2. Pobranie projektu

Projekt ma otwarte źródła i możemy go pobrać z <https://github.com/mimo00/thesis> (rysunek 6.1). Pobrany zostanie kod źródłowy projektu wraz z całą jego historią rozwoju, opis struktury projektu znajduje się 5. *Opis rozwiązań*

Rysunek 6.1 Widok do pobrania projektu



Źródło: własne

6.3. Generowanie przykładowych danych

Po pobraniu i rozpakowaniu w docelowej lokalizacji należy uruchomić skrypt inicjalizujący bazę danych przykładowymi danymi:

```
./.../thesis/scripts/run-init
```

⁴ cURL - Program służący do wysyłania zapytań HTTP <https://curl.haxx.se/>

Podczas generowania danych zostało wygenerowane konto administratora systemu o loginie: *root* i hasłem *root*

6.4. Uruchomienie projektu

Uruchamiamy aplikację poleceniem wykonanym z poziomu terminala:

```
docker-compose up
```

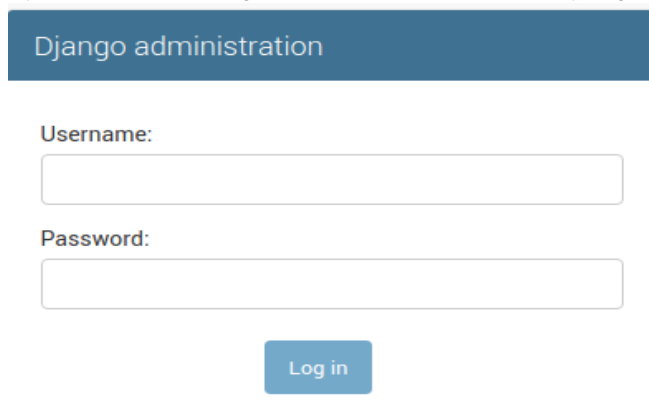
W celu sprawdzenia, czy uruchomienie aplikacji powiodło się, należy odwiedzić link: <http://localhost:8000>

6.5. Utworzenie konta użytkownika

W celu przystąpienia do pracy z systemem jako administrator, musimy posiadać konto z prawami administracyjnymi.

W celu utworzenia nowego użytkownika, musimy utworzyć jego konto (p.u. A01). Aby to zrobić, należy zalogować się do panelu administracyjnego pod adresem: <http://localhost:8000/admin> tak jak na rysunku 6.2

Rysunek 6.2 Widok logowania do panelu administracyjnego



Źródło: własne

Po zalogowaniu widzimy, jak na rysunku 6.3 menu główne, wybieramy opcję dodania nowego użytkownika do systemu, a następnie wprowadzamy dane użytkownika: nazwę i hasło.

Rysunek 6.3 Widok panelu administracyjnego z wyszczególnieniem dodania użytkownika

AUTH TOKEN		
Tokens	+ Add	Change
AUTHENTICATION AND AUTHORIZATION		
Users	+ Add	Change
DECISIONS		
Aggregator decisions	+ Add	Change
Aggregator group decisions	+ Add	Change
Schedule decisions	+ Add	Change
SCHEDULES		
Charging points	+ Add	Change
Electric vehicles	+ Add	Change
Nodes	+ Add	Change
Point schedules	+ Add	Change
Schedules	+ Add	Change

Źródło: własne

6.6. Wprowadzenie węzła, punktu ładowania, samochodu elektrycznego

W analogiczny sposób, podając potrzebne dane możemy dodać samochód elektryczny (p.u. A04), węzeł (p.u. A02) i punktu ładowania (p.u. A03). Przykład znajdziemy na rysunku 6.4

Rysunek 6.4 Widok panelu administracyjnego, dodanie encji

AUTH TOKEN		
Tokens	+ Add	Change
AUTHENTICATION AND AUTHORIZATION		
Users	+ Add	Change
DECISIONS		
Aggregator decisions	+ Add	Change
Aggregator group decisions	+ Add	Change
Schedule decisions	+ Add	Change
SCHEDULES		
Charging points	+ Add	Change
Electric vehicles	+ Add	Change
Nodes	+ Add	Change
Point schedules	+ Add	Change
Schedules	+ Add	Change

Źródło: własne

6.7. Złożenie planu ładowania

Gdy użytkownik istnieje, może on złożyć swój plan ładowania pojazdu na kolejny dzień (p.u. U01). Aby to zrobić musi najpierw uzyskać token który następnie będzie wysyłał wraz z zadaniami w celu uwierzytelnienia i autoryzacji, dokumentacja interfejsu w rozdziale 5.4.1. *Zarządzanie tokenami*

W celu uzyskania tokena:

```
curl --data "username=nazwa" --data "password=haslo" -X POST 127.0.0.1:8000/api-token-auth/
```

W miejscu nazwy i hasła wpisujemy dane podane przy tworzeniu użytkownika.

W odpowiedzi dostaniemy token np:

```
{"token": "c0a7456526025babd6cbc83fe39f7e29ee644ead"}
```

W celu wysłania planu ładowania należy skorzystać z interfejsu `/charging-schedules/`. Szczegóły interfejsu sieciowego przedstawiono w 5.4.2. *Składanie planów ładowania*. Poniżej zostało przedstawione żądanie.

```

curl -H "Authorization: Token c0a7456526025babd6cbc83fe39f7e29ee644ead"
-H "Content-Type: application/json"
-d '{
    "electric_vehicle": 1,
    "mode": "hh",
    "charge_percent": 10,
    "trip_percent": 60,
    "point_schedules": [
        {
            "point": 1,
            "arrival_time": "2019-03-10T06:30:00",
            "departure_time": "2019-03-10T12:30:00",
        },
        {
            "point": 1,
            "arrival_time": "2019-03-10T20:30:00",
            "departure_time": "2019-03-10T23:59:00",
        }
    ]
}'
-H "Content-Type: application/json" 127.0.0.1:8000/charging-schedules/

```

Po złożeniu planu ładowania przez użytkownika, administrator może zobaczyć ją w panelu administracyjnym (p.u. A05). Rysunek 6.5, rysunek 6.6, rysunek 6.7 przedstawiają instrukcję jak zobaczyć szczegóły planu ładowania.

Rysunek 6.5 Widok panelu administracyjnego, listy planów ładowania

AUTH TOKEN		
Tokens	+ Add	Change
AUTHENTICATION AND AUTHORIZATION		
Users	+ Add	Change
DECISIONS		
Aggregator decisions	+ Add	Change
Aggregator group decisions	+ Add	Change
Schedule decisions	+ Add	Change
SCHEDULES		
Charging points	+ Add	Change
Electric vehicles	+ Add	Change
Nodes	+ Add	Change
Point schedules	+ Add	Change
Schedules	+ Add	Change

Źródło: własne

Rysunek 6.6 Widok listy odebranych planów ładowania

Django administration			
Home > Schedules > Schedules			
Select schedule to change			
Action: <input type="text" value="-----"/> <input type="button" value="Go"/> 0 of 75 selected			
<input type="checkbox"/>	ID	DATE	MODE
<input type="checkbox"/>	76	May 8, 2019, 4:48 p.m.	home-home
<input type="checkbox"/>	75	May 4, 2019, 4 p.m.	home-work-home
<input type="checkbox"/>	74	May 4, 2019, 4 p.m.	home-work-home
<input type="checkbox"/>	73	May 4, 2019, 4 p.m.	home-work-home
<input type="checkbox"/>	72	May 4, 2019, 4 p.m.	home-work-home
<input type="checkbox"/>	71	May 4, 2019, 4 p.m.	home-work-home
<input type="checkbox"/>	70	May 4, 2019, 4 p.m.	home-work-home

Źródło: własne

Rysunek 6.7 Widok szczegółowy planu ładowania

Change schedule

Mode: home-work-home

Charge percent: 50

Trip percent: 60

Electric vehicle: ElectricVehicle #121 owner foreman

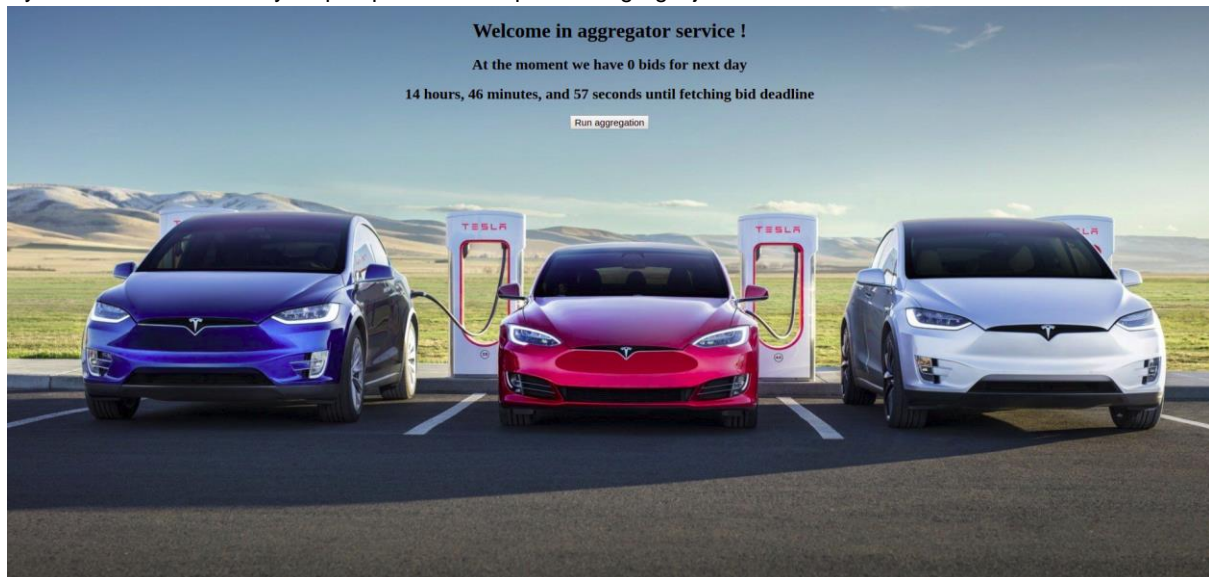
POINT SCHEDULES		
ARRIVAL TIME	DEPARTURE TIME	POINT
Schedule #1813 schedule 605		
Date: 2019-05-22 Today	Date: 2019-05-22 Today	ChargingPoint object (2)
Time: 00:00:09 Now	Time: 07:17:09 Now	
Note: You are 2 hours ahead of server time.		
Schedule #1814 schedule 605		
Date: 2019-05-22 Today	Date: 2019-05-22 Today	ChargingPoint object (3)
Time: 08:00:09 Now	Time: 15:19:09 Now	
Note: You are 2 hours ahead of server time.		
Schedule #1815 schedule 605		
Date: 2019-05-22 Today	Date: 2019-05-22 Today	ChargingPoint object (2)
Time: 18:00:09 Now	Time: 23:16:09 Now	
Note: You are 2 hours ahead of server time.		

Źródło: własne

6.8. Uruchomienie procesu agregacji

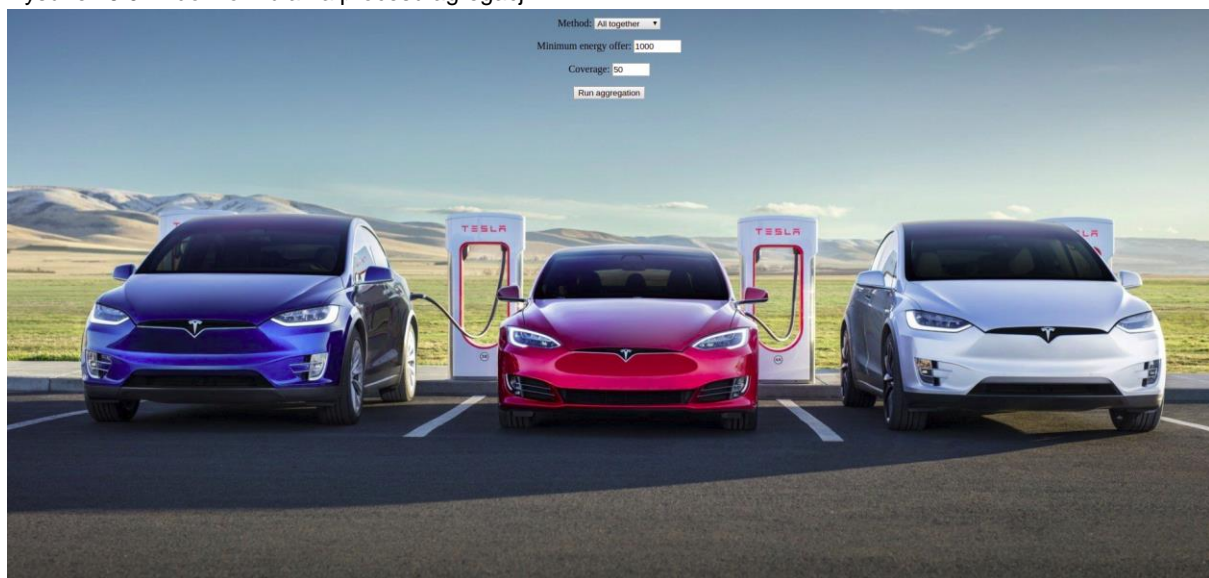
Po zebraniu ofert administrator może uruchomić proces agregacji (p.u. A07) na stronie <http://127.0.0.1:8000/auction> klikając przycisk *Run Aggregation* tak jak na rysunku 6.8, a następnie wprowadza on parametry procesu agregacji rysunek 6.9, po przeprowadzonym procesie agregacji administrator zobaczy szczegóły procesu przykład na rysunku 6.10.

Rysunek 6.8 Widok strony do przeprowadzenia procesu agregacji



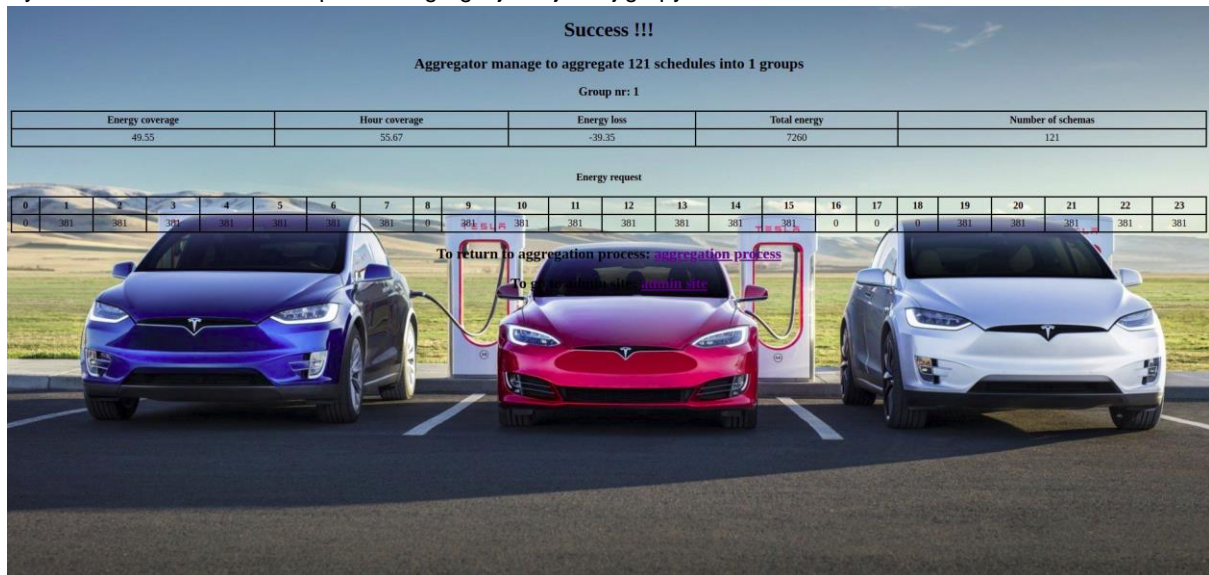
Źródło: własne

Rysunek 6.9 Widok formularza procesu agregacji



Źródło: własne

Rysunek 6.10 Widok rezultat procesu agregacji dla jednej grupy



Źródło: własne

6.9. Odebranie planu ładowania

Po przeprowadzonym procesie agregacji, po którym zostaną wygenerowane plany ładowania dla samochodu, użytkownik może pobrać zatwierdzone plany. Szczegóły interfejsu sieciowego przedstawiono w 5.4.3. *Udostępnianie decyzji*

curl

```
-H "Authorization: Token c0a7456526025babd6cbc83fe39f7e29ee644ead"
-X GET 127.0.0.1:8000/charging-localization-decision/?electric_vehicle=1
```


7. Testy

7.1. Testy automatyczne aplikacji

7.1.1. Wykorzystane narzędzia

Do testowania aplikacji posłużono się bibliotekami:

- pytest-django
- factory_boy
- unittest

7.1.2. Uruchomienie

W celu uruchomienia automatycznego testu aplikacji, należy uruchomić skrypt poleceniem:

```
./scripts/run-tests
```

Skrypt ten utworzy nową bazę danych, z której wszystkie dane po zakończeniu testów zostaną usunięte.

7.1.3. Zakres

W testowaniu aplikacja skupiono się na testach jednostkowych i integracyjnych. Testy jednostkowe testują pojedynczą część kodu taką jak funkcje czy klasy. W projekcie można znaleźć jednostkowe testy serializacji i deserializacji danych do formatu zrozumiałego przez agregator. Testy znajdują się w plikach:

- `/thesis/thesis/apps/aggregator_integration/tests/test_generator.py`
- `/thesis/thesis/apps/aggregator_integration/tests/test_loader.py`
- `/thesis/thesis/apps/aggregator_integration/tests/test_time_utils.py`

Testy integracyjne testują współdziałanie kilku komponentów systemu na raz, w ten sposób testowano w projekcie interfejsy sieciowe. Testując interfejsy sieciowe, testujemy komponent takie jak: autoryzację, logikę biznesową, walidację danych. Testy interfejsów sieciowych możemy znaleźć w plikach:

- `/thesis/thesis/apps/schedules/tests/test_views.py`
- `/thesis/thesis/apps/decisions/tests/test_views.py`

7.2. Testy wydajnościowe

Do testów wydajnościowych aplikacji posłużono się narzędziem *wrk*⁵. Jest to narzędzie umożliwiające wygenerowanie pewnego ruchu sieciowego i monitorujące czasy odpowiedzi. Testy zostały przeprowadzone przy użyciu 10 równoległych wątków pracujących przez 30 sekund dla każdego przypadku testowego. Tabela 7.1 przedstawia wyniki testu wydajności dla interfejsów pobierania planów ładowania

⁵ Źródła projektu: <https://github.com/wg/wrk>

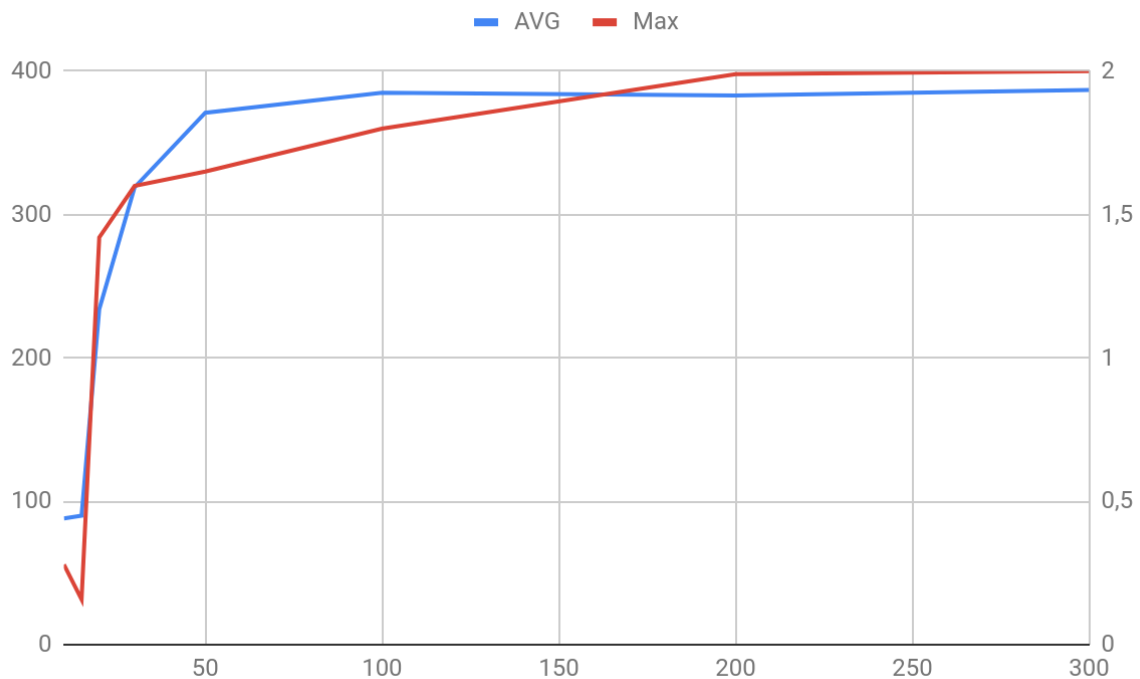
przez samochody elektryczne. Parametrem, który zmieniano podczas testów, jest ilość podtrzymywanych połączeń w czasie żądania C. Badanymi miarami są średni (AVG) i maksymalny (MAX) czas odpowiedzi podczas przeprowadzanych testów. Na wykresie 7.1 przedstawiono dane z tabeli numer 7.1.

Tabela 7.1 Tabela testów wydajności dla pobierania planów ładowania

C	AVG	MAX
10	88	0,279
15	90	0,158
20	234	1,42
30	319	1,6
50	371	1,65
100	385	1,8
200	383	1,99
300	387	2

Źródło: Własne

Wykres 7.1 Wykres testów wydajności dla pobierania planów ładowania



Źródło: Własne

Testy wydajności pokazały, że aplikacja spełnia założenia o wymaganej przepustowości aplikacji.

8. Zakończenie

W ramach pracy stworzono aplikację internetową wspierającą agregator, umożliwiającą składanie i odbieranie planów ładowania przez samochody elektryczne i zarządzanie procesami agregacji i dezagregacji przed administratorem aplikacji. Aplikacja umożliwia zintegrowanie wcześniej opracowanych algorytmów agregacji i dezagregacji i integruje się z przykładową implementacją, zawiera też testy automatyczne potwierdzające pokrycie przez nią wymagań postawionych przed rozpoczęciem projektu. Aplikacja jest dostępna publicznie i jest gotowa do pobrania i użycia a dzięki konteneryzacji jest bardzo łatwo przenośna. Dzięki popularnemu protokołowi przesyłu informacji JSON umożliwia łatwą implementację klientów aplikacji w dowolnej technologii. Aplikację wykonano w oparciu o darmowe frameworki z otwartymi źródłami co umożliwia jej bardzo łatwy rozwój. Technologie wybrane w projekcie są wiodącymi na rynku frameworków internetowych co w przyszłości pozwoli uniknąć długu technologicznego.

Aplikacja stworzona w ramach tej pracy jest częścią docelowego produktu, który mógłby być używany komercyjnie i zawiera pewne uproszczenia, które przy dalszym rozwoju należałoby zmienić. Aplikacja w obecnej formie wymaga od operatora przeprowadzania procesu agregacji i dezagregacji ręcznie co w środowisku produkcyjnym należałoby zautomatyzować. W obecnej formie, z uwagi na to, że decyzja rynku jest symulowana, proces agregacji ofert sprzedaży i dezagregacji są przeprowadzane synchroniczne. W ostatecznej wersji aplikacji należałoby wprowadzić mechanizmy wspierające asynchroniczność tego procesu. W środowisku produkcyjnym należałoby rozbudować zarządzanie procesem agregacji, ponieważ w aktualnej wersji zarządzanie jest znacznie uproszczone.

Planując dalszy rozwój produktu należałoby zaimplementować klientów aplikacji w urządzeniu ładującym. Codzienne wprowadzanie danych może być dla użytkownika uciążliwe więc ciekawą perspektywą jest wykorzystanie sztucznej inteligencji, która uczyłaby się zachowań użytkownika i na tej podstawie składała plany ładowania za niego. Użytkownik po otrzymaniu planów ładowania nie powinien być odpowiedzialny za ładowanie w odpowiednich chwilach, potrzebowałby on modułu odpowiadającego za pobranie energii zgodnie z zatwierdzonym planem. Zautomatyzowanie mechanicznych czynności spowodowałoby obniżenie narzutu związanego z korzystaniem z usług agregatora, które ograniczyłoby się tylko do rejestracji do wybranego produktu na rynku.

9. Literatura

[1] Jean-Michel Clairand, Javier Rodríguez-García i Carlos Álvarez-Bel, *Smart Charging for Electric Vehicle Aggregators Considering Users' Preferences*, October 2018

[2] Karolina Drabarz, *Projekt i implementacja algorytmów agregacji i dezagregacji ofert pojazdów elektrycznych*, Warszawa 2019

[3] Wymagania funkcjonalne, przypadki użycia
http://www.cs.put.poznan.pl/jrojek/files/io1/requirements/Wymagania_opis.pdf

[4] Krzysztof Sacha, *Inżynieria oprogramowania* Wydawnictwo naukowe PWN Warszawa 2010

[5] Crockford D., *The application/json Media Type for JavaScript Object Notation (JSON)*, RFC 4627, 2006

10. Spis rysunków

1.1 Liczba samochodów elektrycznych w latach 2013-2017	9
1.2 Schemat scentralizowanego zarządzania flotą pojazdów elektrycznych	10
1.3 Diagram przypadków użycia	14
5.1 Schemat UML relacji encji	22
6.1 Widok do pobrania projektu	32
6.2 Widok logowania do panelu administracyjnego	33
6.3 Widok panelu administracyjnego z wyszczególnieniem dodania użytkownika	34
6.4 Widok panelu administracyjnego, dodanie encji	35
6.5 Widok panelu administracyjnego, listy planów ładowania	37
6.6 Widok listy odebranych planów ładowania	37
6.7 Widok szczegółowy planu ładowania	38
6.8 Widok strony do przeprowadzenia procesu agregacji	39
6.9 Widok formularza procesu agregacji	39
6.10 Widok rezultat procesu agregacji dla jednej grupy	40
7.1 Wykres testów wydajności dla pobierania planów ładowania	42

11. Spis tabel

5.1 Tabela odpowiedzi, interfejs do składania planów ładowania	25
5.2 Tabela odpowiedzi, interfejs do pobierania planów ładowania	27
7.1 Tabela testów wydajności dla pobierania planów ładowania	42