Dokumentacja Techniczna

Projekt:

Przeprowadzenie prac badawczych i rozwojowych umożliwiających wdrożenie inteligentnego kontraktu opartego o technologię blockchain

Kontekst:

wymagania FURPS+, dokumentacja techniczna z wyników analiz, diagramy projektowe schematy blokowe ze szczegółami nt. elementów systemu. Szczegóły systemu, opis metod i funkcjonalności.

Autorzy:

- Blicharski Bartłomiej
- Martin Morawiec
- Tomas Sliwka

Fundusze Europejskie Program Regionalny | Rzeczpospolita Polska | Śląskie. | Unia Europejska Europejski Fundusz Rozwoju Regionalnego

Spis treści

I. FURPS

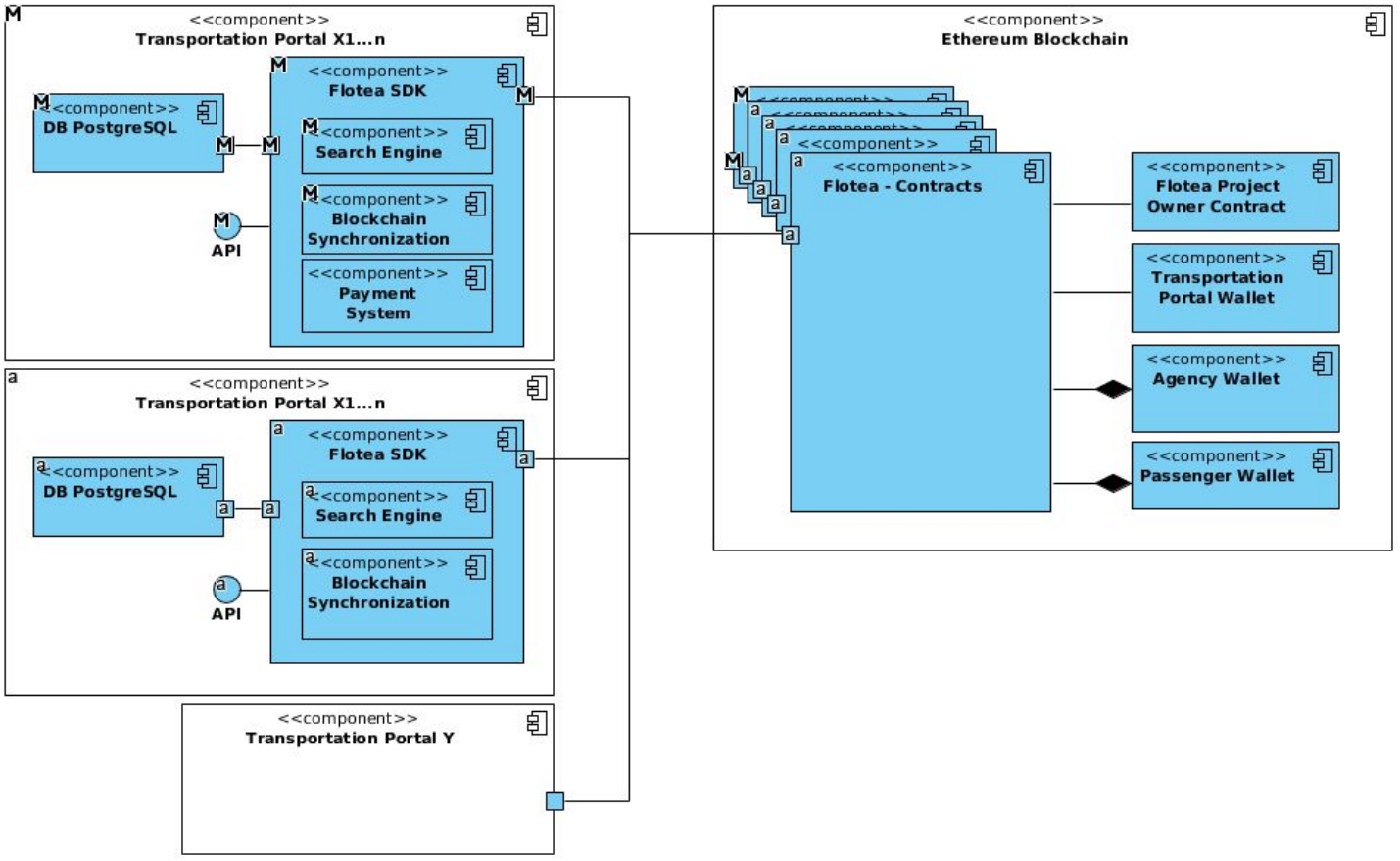
Podstawowa tablica wymagań FURPS+. Podstawowe wymagania, które system musi spełnić po wykonaniu projektu posortowane wg. konkretnych wartości. Poniższy FURPS+ dotyczy podstawowych wymagań względem oprogramowania, został zmodyfikowany podczas projektu.

Funkcjonalne	Podstawowe 1. Synchronizacja bazy danych z połączeniami z Blockchain 2. Obsługa własnego Tokenu 3. Konsensus przewoźników Wyszukiwarka: 1. Wyszukiwanie wg. radiusa 2. Wyszukiwanie po współrzędnych typu box 3. Dostęp do szczegółowych danych połączenia 4. Parametry wyszukiwarki takie jak data, czas, czy obsługuje osoby z niepełnosprawnościami Zamieszczanie tras: 1. Zamieszczanie trasy zwykłej 2. Zamieszczanie trasy w połączeniu 1 z 2 3. Mechanizm zamieszczenia wytworzonej trasy do Blockchain 4. Mechanizm harmonogramowania Rezerwacje: 1. Obsługa miejsc w pojazdach (sprzedaż miejsc) 2. System zakupu Tokenów 3. Możliwość sprzedaży miejsc przez agenta
Użyteczności	1. Obsługa połączeń typu door 2 door (ze wskazaniem miejsca odbioru i dostarczenia) 2. Obsługa połączeń lotniczych 3. Obsługa połączeń morskich 4. Obsługa połączeń linii kolejowych 5. Obsługa połączeń innych środków transportu 6. Możliwość łatwego podłączenia pod dowolny serwis do wyszukiwania połączeń
Niezawodność	1. Testy obciążeniowe

	2. Możliwość przeszukiwania tysięcy tras
Wydajność	1. Lokalna baza danych połączeń 2. Połączenia wielokrotne zgodne ze schematami POI DataProvider (scheduler) 3. Kompaktowość, kontrakt i POI nie może zawierać wielu informacji poza tymi które są potrzebne. 4. Wyszukiwanie nie może trwać długo (niedopuszczalne są czasy powyżej maksymalnie kilku sekund (do 5 sekund przy sporym obciążeniu).
Wspieralność	1. Obsługa standardów GTFS z FIWARE DataModels w strukturach danych 2. API REST w oparciu o JSON
Implementacja	1. Instrukcja wdrożenia i instalacji 2. Wykrywanie i aktualizowanie oprogramowania
Interfejsy	1. SDK w Go Lang - aplikacja kliencka z API 2. (Opcjonalne) zastosowanie MQTT do wymiany informacji

II. Analiza Techniczna

Widok kooperacji aplikacji



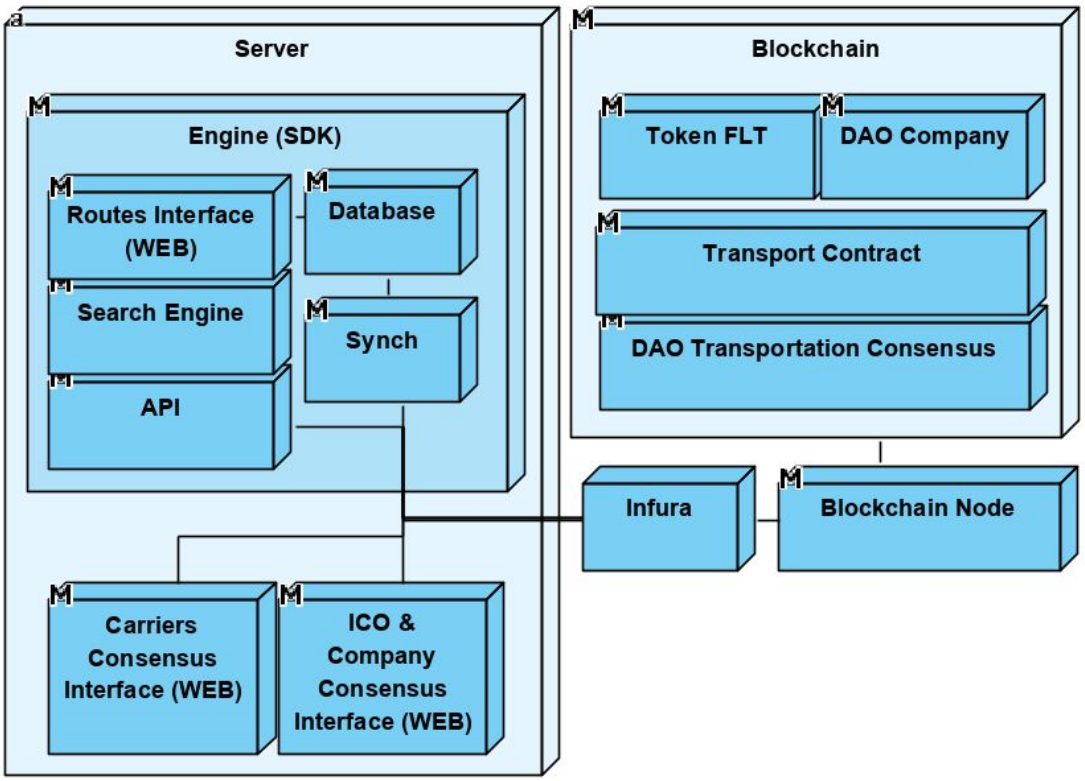
Powyższy widok kooperacji przedstawia ogólny widok dla proponowanego rozwiązanie. Kontrakt Flotea będzie umieszczony w sieci Ethereum Blockchain. Flotea udostępnia narzędzia SDK, które będą wgrywane przez dowolny portal wyświetlający, sprzedający i pośredniczący w sprzedaży transportu z punktu A do punktu B. Narzędzia będą wymagały wypełnienia pliku konfiguracyjnego np. Dostępu do bazy danych. Instalacja narzędzi ma być prosta i uniwersalna, dlatego zostaną zastosowane standardy FI-WARE POI DataProvider oraz standardy FIWARE-Data models dla transportu (GTFS).

Jak widać na powyższym schemacie - Komponent Ethereum Blockchain dotyczy części wspólnej projektu - kontraktów uruchomionych w publicznej sieci Blockchain, które udostępniają swój interfejs (przez linię poleceń lub technologie takie jak web3). Z kontraktów tych - mogą korzystać dowolne portale związane z transportem, pod warunkiem, że zostaną dodani do Konsensusu Przewoźników (czyli umowy ogólnej użytkowników kontraktu, którzy są przewoźnikami i chcą dodawać trasy). Na diagramie widać przykład dwóch portali - każdy z nich posiada uruchomioną kopię narzędzi przygotowanych w ramach projektu. Narzędzia te posiadają wytworzoną komunikację z kontraktami, interfejsy, API, wyszukiwarkę połączeń, synchronizację, lokalną bazę danych potrzebną do wydajnego wyszukiwania i synchronizacji. Posiadają także narzędzia interfejsu ICO, Interfejs konsensusu przewoźników. Wszystkie te narzędzia, każdy z portali może wdrożyć w dowolny sposób, interfejsy mogą się różnić. Istotnym elementem, który je łączy jest wspólna zdecentralizowana przestrzeń na trasy oraz możliwość udostępniania na zasadach B2B połączeń i sprzedaży biletów.

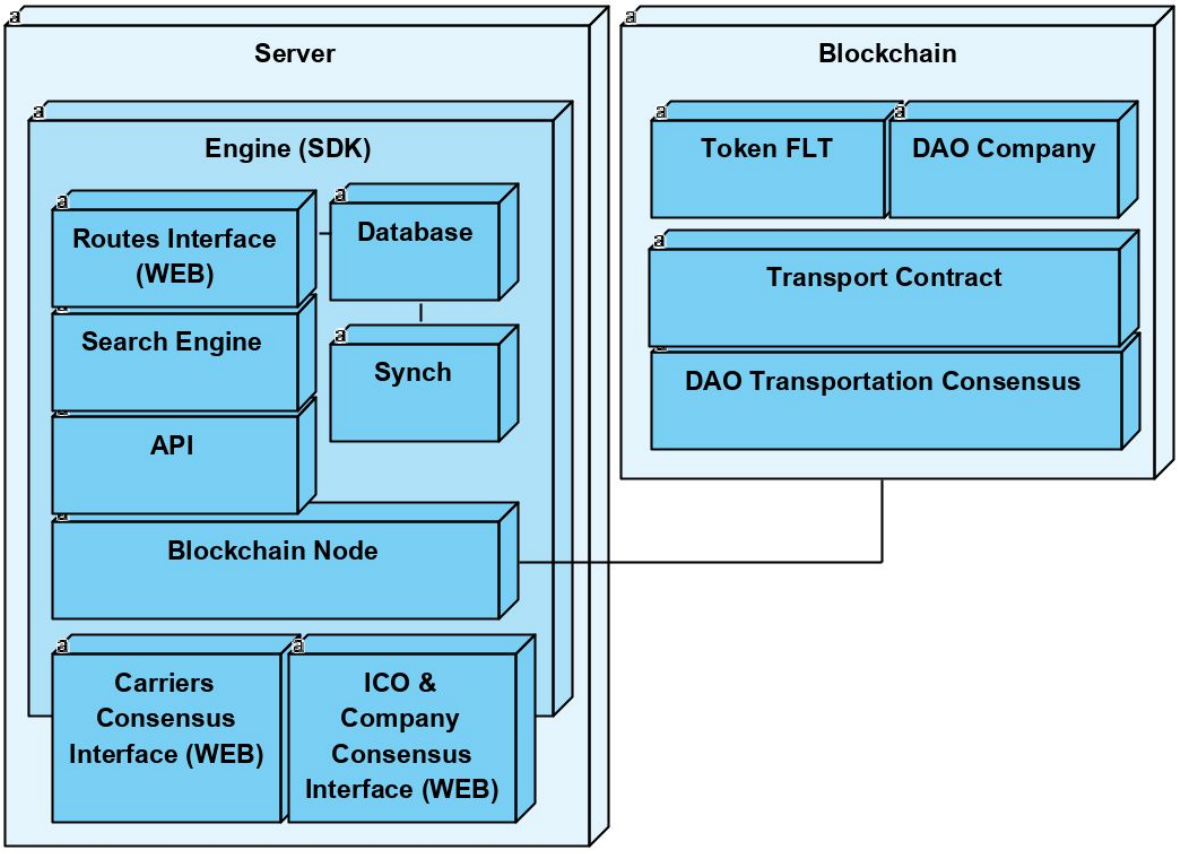
Widok komponentów systemu

Poniższy schemat komponentów przedstawia w jaki sposób system jest skonstruowany. Jak wspomniano wcześniej część systemu znajduje się w formie inteligentnych kontraktów umieszczonych bezpośrednio w Blockchainie Ethereum, nastomiast częśc serwerowych może być wiele (w zależności od ilości wdrożeń). W Blockchain znajdują się Token FLT, DAO Company - firmy, która jako pierwsza wdroży kontrakty, Transport Contract - baza wszystkich ofert tras, DAO Transportation Consensus - Konsensus przewoźników. W obecnej wersji synchronizacja dodanych tras jest wykonywana poprzez integrację z serwisem Infura, która posiada API do komunikacji z Blockchainem, który lokalnie jest aktualizowany w Infura[1].

[1] https://infura.io



W przypadku bardziej zaawansowanego wdrożenia, gdzie istnieje konieczność odizolowania się od zewnętrznych usług typu Infura, istnieje możliwość uruchomienia Blockchain Node oraz połączenie synchronizacji wykonywanej lokalnie (należy pamiętać, że to wymagające rozwiązanie ze względu na to, że należy uruchomić Blockchain Node, kopię Blockchaina na własnym serwerze, co wymaga sporej ilości miejsca). Benefitem takiego rozwiązania jest szybkość komunikacji - w przypadku kiedy serwer na którym wdraża się rozwiązanie posiada dobre łącze z Internetem. Na poniższym diagramie przedstawiono różnicę w tym zakresie. Jak widać połączenie informacyjne pomiędzy Blockchain, a serwerem jest bezpośrednia, ponieważ w takim wypadku Blockchain Node musi być uruchomiony równolegle na serwerze.

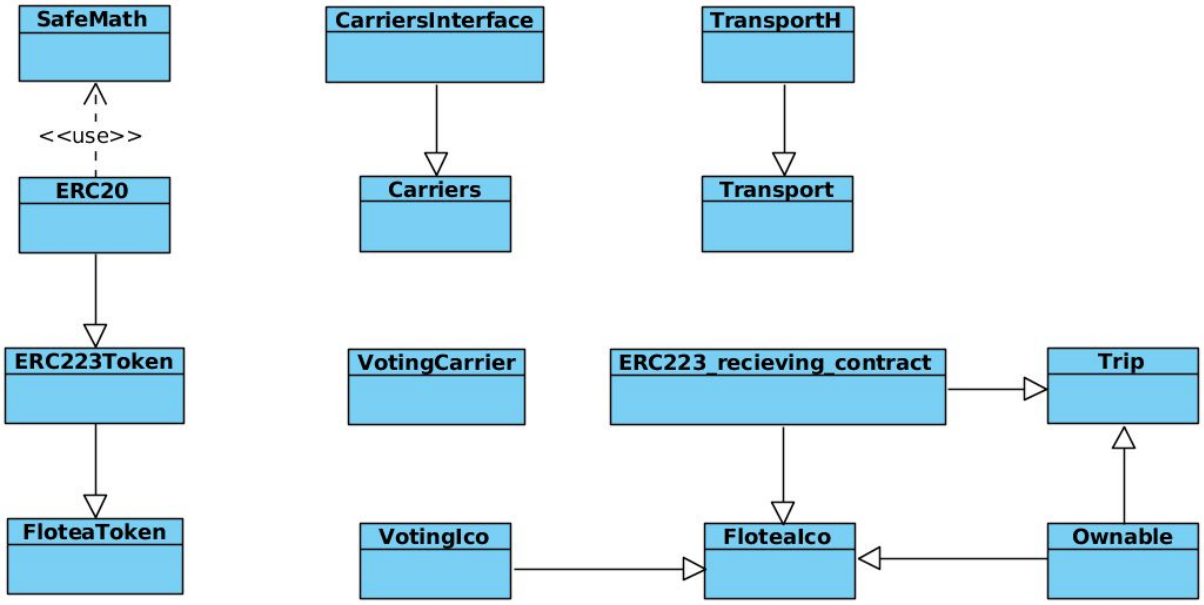


Flotea Kontrakty

Na poniższym schemacie pokazano przegląd wszystkich inteligentnych kontraktów potrzebnych do funkcjonowania całego projektu. Zawiera części potrzebne do wytworzenia własnego tokena ERC20 o nazwie FLT, jego dystrybucję poprzez mechanizmy ICO oraz kontakty, których się używa dla zakupu biletów. Dalej zawiera kontakty dla obsługi tras i system głosowania na przewoźników, którzy mieliby stać się częścią konsensusu, zawiera również głosowanie dla uczestników właścicieli kontraktu Flotea.

Wskazówka: ponieważ technologia web3.js nie umożliwia odbioru tekstowej informacji o błędzie, który wywoływany jest w kontrakcie poprzez funkcję *revert()*, lub *require()*, kontrakty obsługują kilka funkcji z nazwą *before...()*, którą da się uruchomić (metoda call) przed samym wykonaniem transakcji (send), pomaga to w uzyskaniu

bezpłatnych informacji czy da się daną transakcję wykonać bez problemu oraz uzyskać tekstowy opis błędu.



Krótki opis kontraktów

SafeMath - Zawiera metody dla bezpiecznej pracy z dużymi wartościami numerycznymi (dodawanie, odejmowanie, dzielenie i mnożenie).

ERC20 - Standard podstawowego tokenu, operacja na manipulacji z tokenami i jego parametry (nazwa, maksymalna ilość, liczba miejsc po przecinku).

ERC223Token - Standard dla tokenu, którym chcemy płacić w innym kontrakcie, zawiera podstawowe operacje do manipulacji z tokenem i uzupełnienie o atrybut daty.

FloteaToken - Token dziedziczy metody i atrybuty poprzednich kontraktów, ustawia parametry tokenu.

CarriersInterface - Nagłówek dla kontraktu Carriers, używa się dla obniżenia wielkości skompilowanych kontraktów, które używają kontrakt Carriers.

Carries - Zbiór przewoźników, ich stanu i tras.

VotingCarrier - Konsensus przewoźników, kontrakt obsługuje mechanizm głosowania dla dodawania i usuwania przewoźników z konsensusu. Tylko przewoźnicy, którzy znajdują się w konsensusie mają możliwość dodawania tras w ramach blockchain.

TransportH - Nagłówek dla kontraktu Transport, zawiera zdarzenia związane z tworzeniem i edycją przewoźnika, tworzeniem i edycją trasy, zakupem i zwrotem biletów.

Transport - Kontrakt zawierający wszystkie trasy, zawiera metody dla tworzenia tras.

ERC223_recieving_contract - Nagłówek dla kontraktów, który przyjmuje płatności w tokenach FLT.

Ownable - zapewnia właściwości dziedziczonych kontraktów, zawiera modyfikator, który zapewnia, że wybrane metody mogą być wywoływane tylko przez właściciela kontraktu.

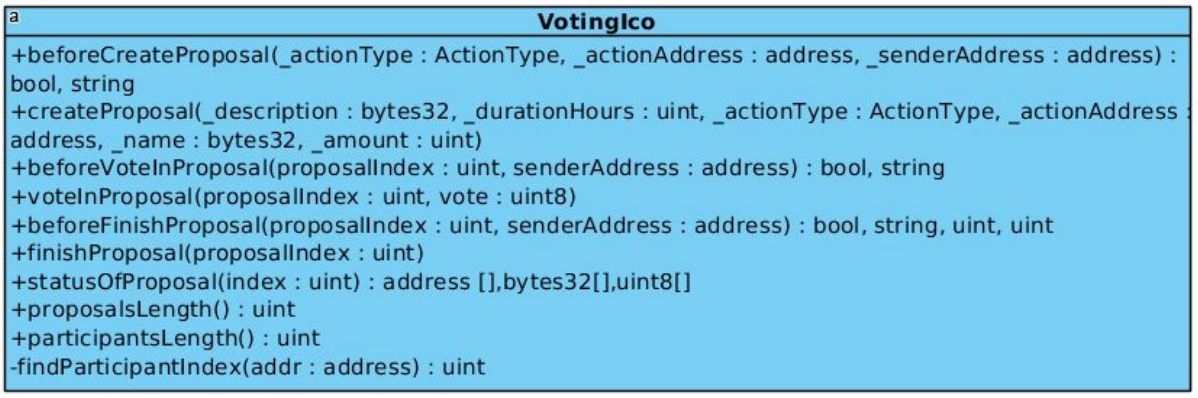
Trip - Zawiera wszystkie informacje o trasie (harmonogram, początkowe i końcowe punkty, opis trasy, typ transportu, ilość miejsc i cena).

FlotealCo - Kontrakt służący do zmiany Ethereum i Polskich Złotych na tokeny FLT.

VotingIco - Konsensus firmy Flotea, metody do głosowania o dodawanie i odbieranie użytkowników, głosowanie o możliwości przesuwania uzyskanych tokenów i Ethereum z ICO.

Flotea Kontrakty - Portfel Kontraktu Flotea

Dla właścicieli, wspólników i pracowników firmy Flotea został wytworzony konsensus na kontrakcie VotingIco, gdzie istnieje możliwość głosowania o dodanie, albo odwrotnie usunięcie właścicieli/wspólników z grupy. Jest również możliwość głosowania o przesunięcie zgromadzonych środków na dalszy rozwój i jest to możliwe w dwóch walutach ETH oraz tokenu FLT.



Powyższy diagram przedstawia strukturę kontraktu VotingIco. Dla wytworzenia nowego głosowania należy utworzyć propozycję transakcji za pomocą funkcji *createProposal()*, z następujacymi parametrami:

- _description - opis głosowania
- _durationHours - jak długo trwa głosowanie
- _actionType - typ głosowania, możliwości są następujące:
  - add_voter - dodanie do konsensusu,
  - remove_voter - usunięcie z konsensusu,
  - transfer_eth - przelew pozyskane ethereum za pomocą ICO.
  - transfer_flt - przesuń tokenów FLT uzyskanych jako prowizje ze sprzedaży biletów.
- _actionAddress - adres portfela, który będzie użyty w przypadku dodania, albo odebrania z konsensusu, albo adres na który będzie przesłany ETH lub FLT.
- _name - nazwa, używa się w przypadku dodawania do konsensusu.
- _amount - ilość, poużywa się przy transferach EtH lub FLT.

Przed tym niż prześlemy taką transakcję, istnieje możliwość skontrolowania czy nie kontrakt nie zwróci jakiegoś błędu. Należy wtedy uruchomić metodę *beforeCreateProposal()*, która zwraca konkretny błąd z tekstowym opisem. Na przykład, kiedy nie jesteśmy członkiem konsensusu, albo kiedy chcemy dodać użytkowników, który już w konsensusie uczestniczy. W przypadku usunięcia

kontroluje się, czy taki użytkownik jest w konsensusie, a także po odebraniu użytkownika liczba użytkowników konsensusu obniży się o jeden.

Do czasu wygaśnięcia głosowania, głosować mogą członkowie konsensusu w konkretnym głosowaniu zawsze, ale tylko raz. Członek konsensusu głosuje za pomocą funkcji voteInProposal(), która ma dwa parametry: proposalIndex (identyfikator głosowania) oraz vote (głos), wartości mogą być:

0 - nie;

1 - tak;

2 - rezygnuję;

Przed głosowaniem sprawdzana jest czy dana transakcja głosowania przejdzie za pomocą wywołania funkcji *beforeVoteInProposal(),* a w przypadku błędu uzyskujemy informację, czy użytkownik jest w konsensusie, czy głosowanie z takim indeksem istnieje, albo czy już się zakończyło, albo czy już głosowanie wcześniej nastąpiło.

Wszystkie głosowania są widoczne i identyfikowalne, można np. Dowiedzieć się na temat stanu głosowania za pomocą funkcji statusOfProposal(), gdzie jedynym wymaganym parametrem jest indeks głosowania. Funkcja zwraca trzy informacje: adres głosowania, identyfikator głosującego i pole głosu. Kontrakt zawiera trzy zdarzenia, które można odebrać i otrzymać aktualne dane:

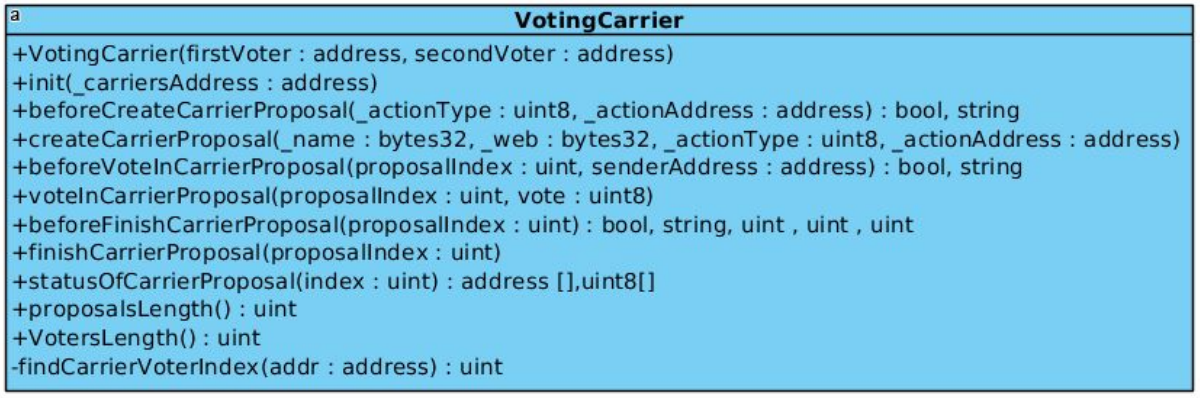
- Vote(bytes32 description, uint proposalIndex, address addr, uint8 vote) - wydarzenie dotyczące głosowania, otrzymamy opis głosowania, indeks, adres głosującego i głos
- FinishVoting(bytes32 description, bool result, uint proposalIndex) - ukończenie głosowania, zwraca opis głosowania, logiczną wartość, tego, czy głosowanie przeszło oraz indeks głosowania
- ProposalCreated(bytes32 description, uint endTime, ActionType actionType, address actionAddress, bytes32 name, uint amount, uint proposalIndex) - zdarzenie podczas tworzenia głosowania, zwraca jej opis, czas zakończenia, typ głosowania, adres do przesłania Tokenu FLT, ETH lub dodanie / usunięcie z konsensusu (także nazwę, ilość oraz indeks).

Z każdego sprzedanego biletu zostaje pobierana prowizja na adres kontraktu Transport - ogólna opłata transakcyjna. Aby można było poprzez konsensus przesłać zebrane tokeny, musi być dozwolone składowanie z tokenu kontraktu Transport na kontrakt VotingIco. Metodą *approve()* pozwolimy na załadowanie do 10^8 tokenów FLT w VotingIco.

Flotea Kontrakty - Portfel Przewoźnika

Aby przewoźnik mógł umieszczać trasy, musi być częścią konsensusu przewoźników. Do tego celu przewoźnik musi zgłosić chęć przystąpienia do konsensusu. Może tego dokonać - tworząc chęć przystąpienia za pomocą funkcji *createCarrierProposal()* w kontrakcie *VotingCarrier*. Funkcja ta wymaga parametrów do identyfikacji przewoźnika: nazwa, adres strony internetowej, dalej typ zgłoszenia (0 - dodanie, 2 - usunięcie) oraz adresy portfeli przewoźnika. Przed tym krokiem da się wywołać funkcję *beforeCreateCarrierProposal()*, która sprawdza czy możliwe jest utworzenie danej propozycji. Funkcja zwraca dwie wartości, wartość typu bool czy wystąpił problem wraz z opisem ciągu błędu. Mogą wystąpić trzy problemy:

1. Kiedy przewoźnik tworzy chęć przystąpienia, a już jest dodany do konsensusu.
2. Kiedy ktoś będzie chciał usunąć przewoźnika, który nie jest dostępny w konsensusie.
3. Kiedy ktoś będzie chciał usunąć przewoźnika, a występują 2 te same wpisy w konsensusie.



Po prawidłowym utworzeniu zgłoszenia przewoźnika, który ma zostać dodany do konsensusu następuje głosowanie. Co się odbywa przy pomocy funkcji *voteInCarrierProposal()*, której parametry to dwa numery, jeden to indeks głosowania, a drugi to głos przewoźnika, który znajduje się już w konsensusie (0 - tak, 2 - nie, 3 - wstrzymuje się od głosu). Każdy przewoźnik istniejący w konsensusie może głosować na konkretną propozycję tylko raz. Przed głosowaniem istnieje możliwość wywołania funkcji *beforeVoteInCarrierProposal()* dla skontrolowania sprawy. Ta funkcja wraca typ *bool* dotyczący tego, czy istnieje jakiś problem oraz ewentualny opis problemu w typie *string.* Mogą istnieć następujące problemy:

- Właściciel portfela nie jest w grupie głosujących,
- Właściciel jest zbanowany (usunięty z grupy głosujących),
- Nie istnieje propozycja z danym indeksem,
- Głosowanie się skończyło,
- Skończył się czas na głosowanie,
- Właściciel portfela już głosował.

Propozycja zostanie przyjęta, dopiero wtedy, kiedy ponad połowa głosujących zgodzi się na dodanie przewoźnika do konsensusu, przy czym wszyscy nie są zobowiązani do głosowania. Głosowanie jest aktualnie skonfigurowane tak, że trwa jedną godzinę, po tym czasie można zakończyć głosowanie funkcją *finishCarrierProposal(),* której jedynym parametrem jest indeks głosowania. Przed przesłaniem transakcji można użyć funkcji testującej *beforeFinishCarrierProposal(),* która umożliwia ewentualne wykrycie następujących błędów:

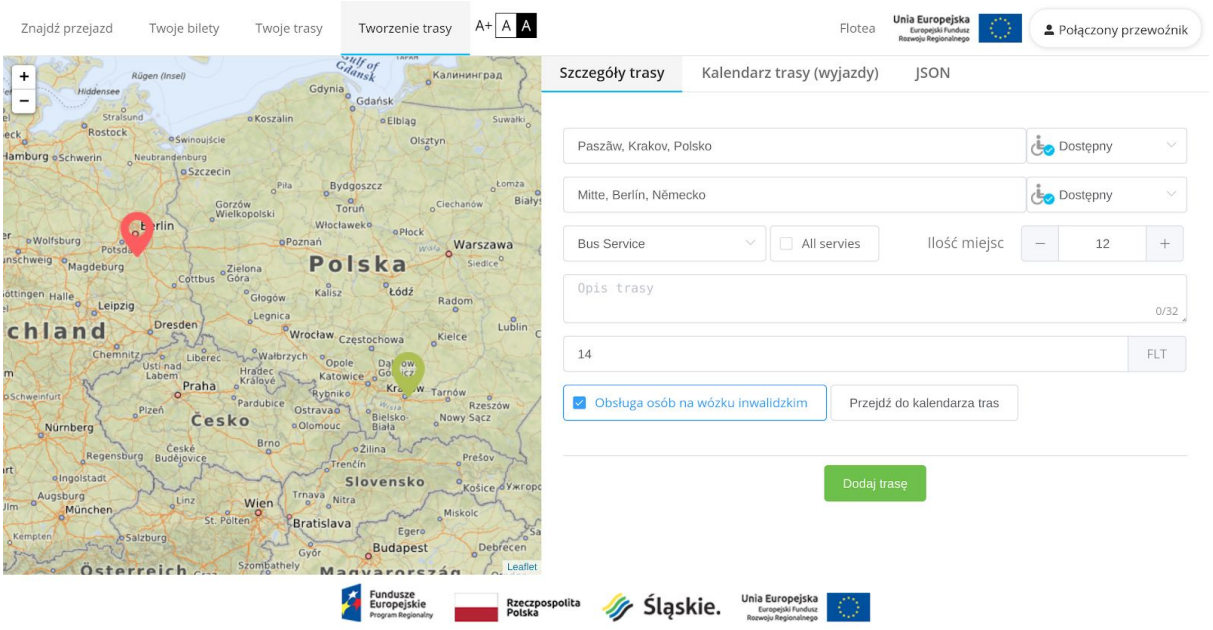
- Propozycja z tym indeksem nie istnieje,
- Głosowanie już było zakończone,
- Głosowanie jest nadal aktywne,
- W przypadku, gdy się głosuje o usunięcie, a aktualna liczba przewoźników w konsensusie jest równa 2, zostanie zwrócona informacja o minimalnej liczbie głosujących.

Kontrakt zawiera metody do uzyskiwania informacji o głosowaniu konsensusu, na przykład wywołaniem *statusOfCarrierProposal()* uzyskamy listę adresów przewoźników i informację o tym w jaki sposób głosowali, wyliczenie propozycji *proposalsLength()* oraz wyliczenie głosujących w konsensusie *VotersLenght()*. Na tym kontrakcie da się subskrybować odbieranie nastepujących zdarzeń:

- VoteCarrier(uint proposalIndex, address addr, uint8 vote) - zdarzenie głosowania, otrzymamy indeks głosowania, adres głosującego i wykonany głos.
- FinishVotingCarrier(bool result, uint proposalIndex, uint votedYes, uint votedNo, uint resigned, uint totalVoters) - zdarzenie zakończenia głosowania, wraca logiczną wartość czy zgłoszenie przeszło, liczbę głosów pozytywnych, negatywnych, ilu głosujących się wstrzymało oraz całkowita liczba głosujących przewoźników z konsensusu.
- CarrierProposalCreated(bytes32 name, bytes32 web, uint endTime, uint8 actionType, address actionAddress, uint proposalIndex) - Zdarzenie przy wytworzeniu głosowania, wraca imię, adres url przewoźnika, czas ukończenia, typ głosowania, adres portfela przewoźnika oraz indeks głosowania.

Flotea Kontrakty - Trasy i harmonogramowanie

Po poprawnym głosowaniu, gdzie przewoźnicy zagłosowali w celu przyjęcia przewoźnika do grupy - w konsensusie pojawia się nowy przewoźnik, który od tej pory może dodawać trasę. Częścią Engine SDK jest aplikacja webowa, która ułatwia dodawanie trasy. Zawiera pola skąd, dokąd przewoźnik jeździ, kompletną listę typu transportu, liczbę miejsc, opis i cenę za przejazd, wyrażoną w tokenach FLT. Poniższy zrzut ekranu prezentuje formularz tworzenia trasy. Jak widać dostępne są funkcjonalności takie jak lista tras, lista biletów, tworzenie trasy, a w nim określenie szczegółów tras, zaawansowany kalendarz tras, zapis harmonogramu do formatu JSON. Dodatkowo podczas dodawania trasy - formularze prezentujące warunki czasowe, które muszą być zastosowane do harmonogramu.



Kolejnym krokiem jest wytworzenie harmonogramu wyjazdów i przyjazdów, zrzut ekranu poniżej przedstawia kalendarz, a w nim zaznaczone czasy, kiedy na danej trasie przewoźnik wyjeżdża. Na jednej trasie może być użytych więcej różnych środków transportu, ale muszą udostępniać minimalną ilość miejsc, która została podana w formularzu tworzenia trasy.

Harmonogram automatycznie zostaje zmieniany do formatu JSON, następnie się kompresuje i jest używany w parametrach wytwarzania trasy. Te kroki są konieczne dla minimalizacji rozmiaru danych, które da się zapisywać w blockchainie, a także z powodu obniżenia ceny za wytworzenie kontraktu typu Trip. Algorytm harmonogramu został napisany w języku JavaScript oraz Go. Jest również używany poprzez funkcję przygotowanego podczas projektu rozszerzenia do bazy danych PostgreSQL i wykorzystywany jako funkcja bazodanowa w formie zapytania.

Poniższy zrzut ekranu prezentuje przykład harmonogramu w formacie JSON:


```
{
  "fromLatLng": [ 50.05250281819231, 19.98892664909363 ],
  "toLatLng": [ 52.520856132425436, 13.406206369400024 ],
  "vehicleType": 42,
  "places": 1,
  "price": "",
  "description": "",
  "schedule": {
    "or": [
      {
        "and": [
          { "wd": [ 3, 6 ] },
          {
            "or": [
              {
                "bev": [ 2020, 4, 24 ],
                "eev": [ 2020, 5, 30, 23, 59, 59 ]
              }
            ]
          },
          {
            "or": [ { "bhr": [ 4 ], "hr": [ 1 ] } ]
          }
        ]
      },
      {
        "and": [
          { "wd": [ 4 ] },
          {
            "or": [
              {
                "bev": [ 2020, 4, 24 ],
                "eev": [ 2020, 5, 23, 23, 59, 59 ]
              }
            ]
          },
          {
            "or": [
              { "bhr": [ 12 ], "hr": [ 1 ] },
              { "bhr": [ 14 ], "hr": [ 1 ] }
            ]
          }
        ]
      }
    ]
  }
}
```

Powyższy harmonogram w JSON opisuje, że w pojeździe na trasę jest jedno wolne miejsce, a trasa odbywa się w każdą środę i sobotę o godzinie 4, w każdy czwartek o godzinie 12 i 14. Trasa trwa godzinę i działa w przedziale czasowym od 24.04.2020 do 30.05.2020. W narzędziach dodawania trasy, można przygotowany w ten sposób JSON modyfikować bezpośrednio, wkleić lub skopiować i przesłać innemu przewoźnikowi.

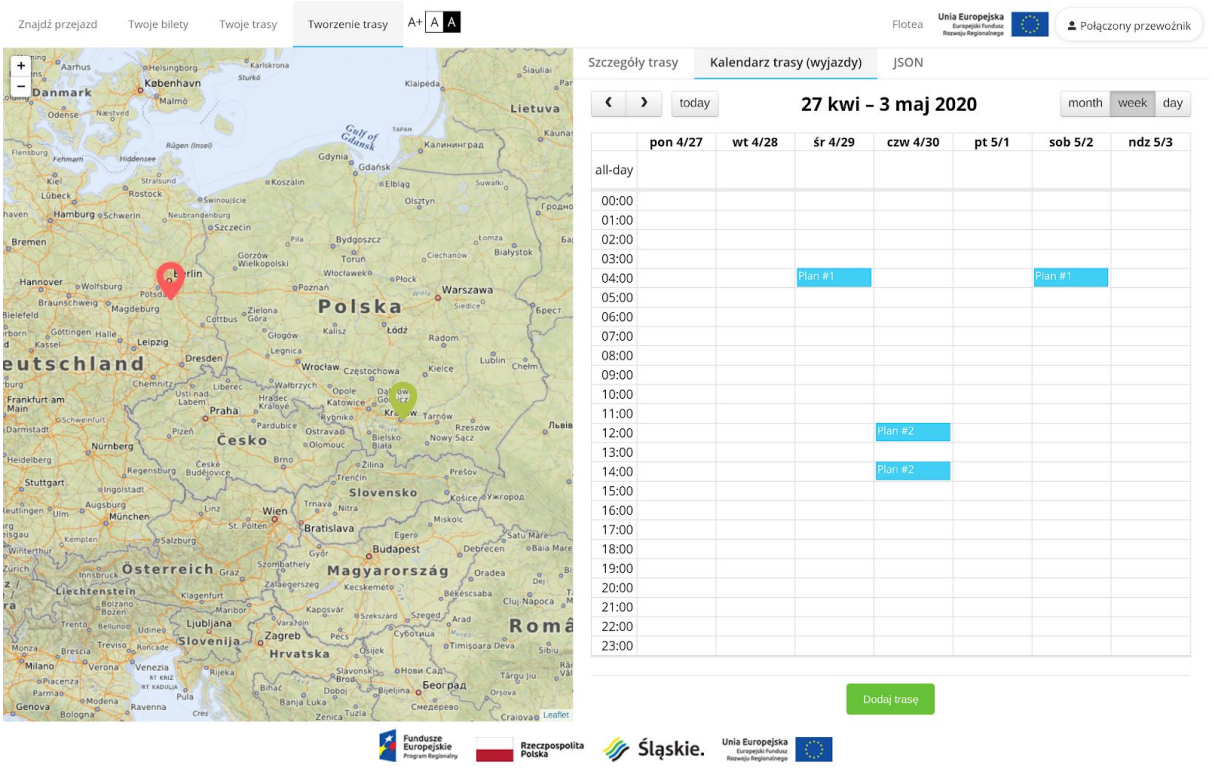
Przyjęty format JSON jest wykonany na podstawie analiz schematów FIWARE POI-DataProvider, który jest udostępniony i dostępny na Licencji wolnego oprogramowania Apache 2.0. Oprogramowanie w tym projekcie różni się od tego zaproponowanego w FIWARE, jest wykonane w innej technologii i w inny sposób, algorytm harmonogramowania, na którym się inspirowano, został usprawniony i przerobiony pod użycie w trasach. Jednakże trzymano się zasad struktury języka opisowego harmonogramu, którego efekt jest widoczny na powyższym screenie. Projekt stosuje schemat, który został szczegółowo opisany w pliku flt/schemas/schdule_schema_3.3.json w projekcie silnika projektu (engine). Tutaj opisane są słowa logiczne, którymi można budować zarówno proste jak i skomplikowane harmonogramy przejazdów w sposób zrozumiały dla człowieka.

Opis poszczególnych słów:

- **or** - unia, jest poprawna kiedy każde z pod harmonogramów spełniają warunki zapytania. Może być tablicą warunków podzieloną na różne struktury. Jeżeli jedna ze struktur się sprawdza, wtedy zapytanie harmonogramu jest poprawne,
- **and** - intersekcja, jest poprawna kiedy wszystkie pod harmonogramy spełniają warunki zapytania. Może być tablicą warunków podzieloną na różne struktury. Zapytanie wykonane może być prawdziwe w momencie, kiedy każdy z warunków AND jest spełniony,
- **not** - zaprzeczenie, jest poprawne kiedy pod harmonogramy nie spełniają warunków zapytania,

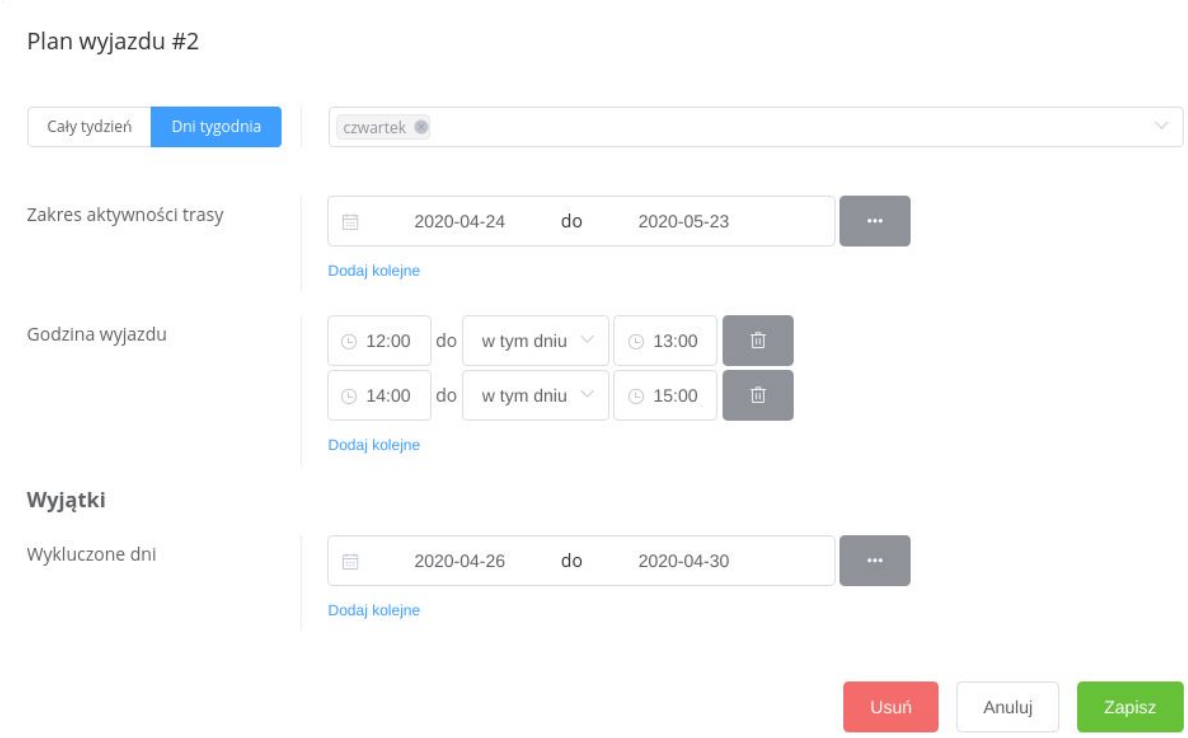
- **wd** - dzień tygodnia, jest poprawny kiedy jest określony w formacie od 1 do 7. Dni tygodnia mogą być tablicą przedstawiającą wyjazdy dostępne w różne dni,
- **bhr** - godzina rozpoczęcia (int godzina, int minuta, int sekunda). Godzin rozpoczęcia może być wiele - może istnieć wiele wyjazdów w danym dni,
- **hr** - ilość godzin liczona od bhr. Może być większa niż 23 godziny.
- **bev** - rozpoczęcie zdarzenia (rok, miesiąc, dzień, godzina, minuta, sekunda). Może przyjąć wartość tablicową (wiele rozpoczęć zdarzenia),
- **eev** - zakończenie zdarzenia (rok, miesiąc, dzień, godzina, minuta, sekunda). Może przyjąć wartość tablicową (wiele rozpoczęć zdarzenia),
- **bdate** - data rozpoczęcia obowiązywania harmonogramu (miesiąc, dzień),
- **edate** - data zakończenia obowiązywania harmonogramu (miesiąc, dzień) liczony od początku roku,

Harmonogram został rozwinięty o możliwość obsługi przejazdów trwających wiele dni, które są popularne np. na długich trasach po Europie. Został dostosowany do potrzeb użycia w transporcie.

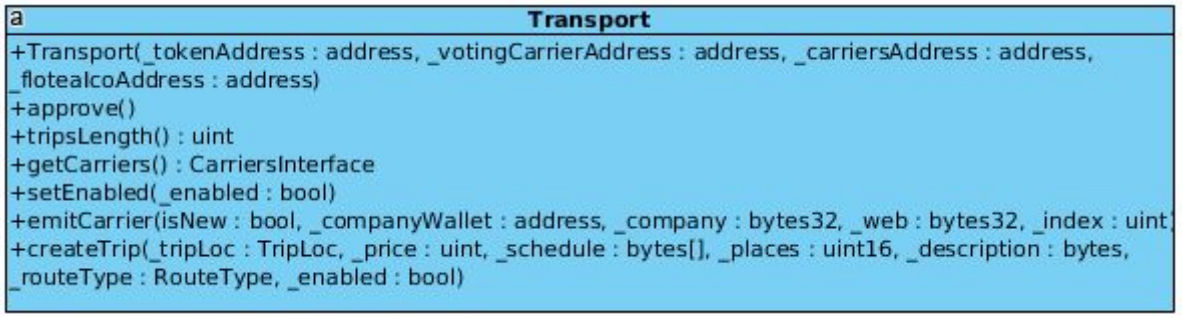


Na powyższym zrzucie ekranu pokazany został kalendarz, w którym wytwarzany jest harmonogram wyjazdu na daną trasę. Kliknięciem (oraz przeciąganiem elementów) wstępnie wytworzymy plan harmonogramu wyjazdów. Formularz składa się z kilku części, najpierw definiujemy czas w którym obowiązuje harmonogram, później tworzymy czasowy harmonogram odjazdów i przyjazdów, przy czym możemy pozwolić przyjazd aż za kilka dni. Następnie harmonogram możemy dalej opisać, dodać ograniczenia na konkretne dni w tygodniu np. Wyjazd tylko w poniedziałki i piątki - tak jak jest zaznaczone na zrzucie ekranu poniżej.

Ponadto na poniższym zrzucie ekranu widać formularz, który posiada wiele funkcjonalności takich jak “dodaj kolejne dni”, “dodaj kolejne godziny”, “dodaj kolejne wyjątki”, czy trasa ma trwać cały tydzień czy w określone dni tygodnia. Jest to interfejs, który nie jest w stanie obsłużyć wszystkich możliwości harmonogramu, ponieważ zbudowanie formularza, który posłużyłby tak rozległe możliwości jest trudna z uwagi na UI/UX.



Poniższy diagram przedstawia kontrakt Transport

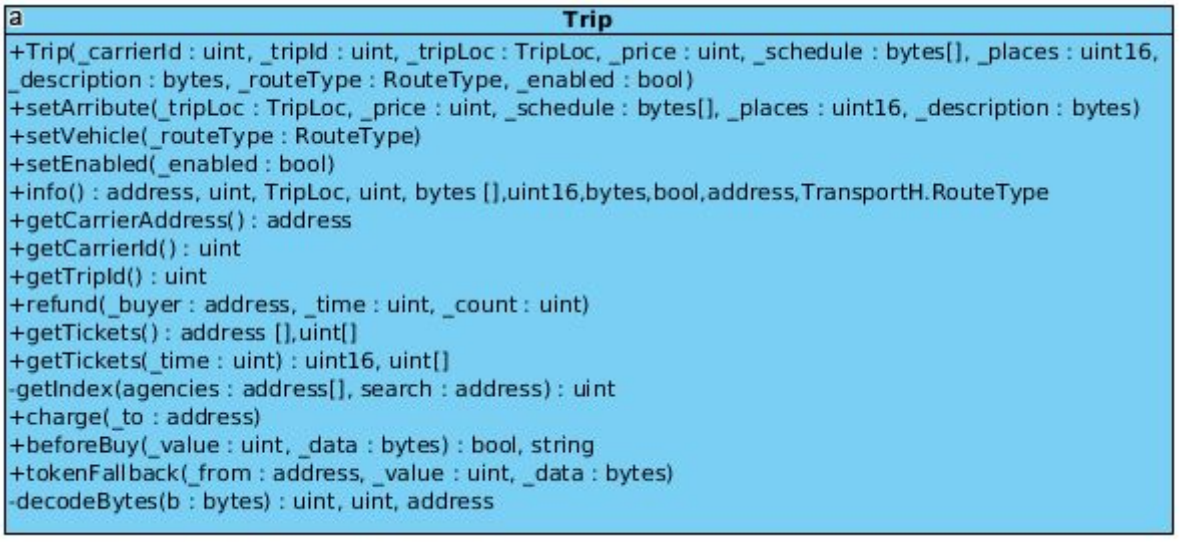


Trasę zapisuje się do blockchain za pomocą funkcji *createTrip(),* z następującymi parametrami:

- _tripLoc - struktura pozycji początku i celu w postaci długości i szerokości geograficznych odzwierciedlających np. Pozycję na mapie *TripLoc {bytes10 fromLat; bytes11 fromLng; bytes10 toLat; bytes11 toLng; }*

- _schedule - skompresowany harmonogram JSON odjazdów i przyjazdów do celu
- _places - liczba miejsc dostępnych do sprzedaży w środku transportowym
- _description - opis trasy
- _routeType - rodzaj transportu
- _enabled - wskazuje czy trasa jest aktywna

Kontrakt dostarcza publiczną metodę dla uzyskania informacji ilości tras *tripsLength()*, której można użyć w momencie kiedy należy poznać adresy wytworzonych tras, można wywoływać zmienną kontraktową jak metodę. Kontrakt zawiera również zmienną trips typu pola struktury *TripStruct { address addr; bool enabled; bool exist; }*. Wywołując metodę trips(1) uzyskamy strukturę drugiej trasy w kolejce w której między innymi przechowywany jest adres trasy *addr*.



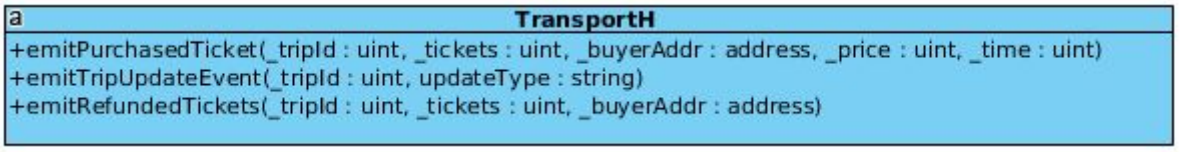
Flotea Kontrakty - Portfel Agencji / Sprzedawcy

SDK może zostać użyte przez każdego kto będzie chciał wyszukiwać trasy i sprzedawać bilety. Parametrem ustawia się adres agenta sprzedaży, a system automatycznie doda tę informację przy sprzedaży. W momencie kiedy trasa będzie

przejechana, może za pomocą metody charge w kontrakcie Trip - zażądać od przewoźnika wypłacenie tokenów FLT. Metoda ta przeniesie tokeny zarówno do konsensusu firmy Flotea jak i prowizje sprzedającemu. 0,1% całkowitej kwoty tokenów zapłaconych przez pasażera w trakcie zakupów biletu. Jeśli bilet zostanie sprzedany przez sprzedającego, otrzyma 10%. Reszta kwoty pozostaje u przewoźnika.

Flotea Kontrakty - Portfel Pasażera

Ktokolwiek może subskrybować odbiór zdarzeń kontraktu TransportH i pozyskiwać w ten sposób informacje o nowym zakupie biletów, dodawaniu trasy, albo otrzymać zwrot za bilet w określonym czasie. Poniżej widzimy definicję zdarzeń oraz zwracane przez nie wartości.



_tripId: identifikator trasy

_tickets: ilość kupionych biletów

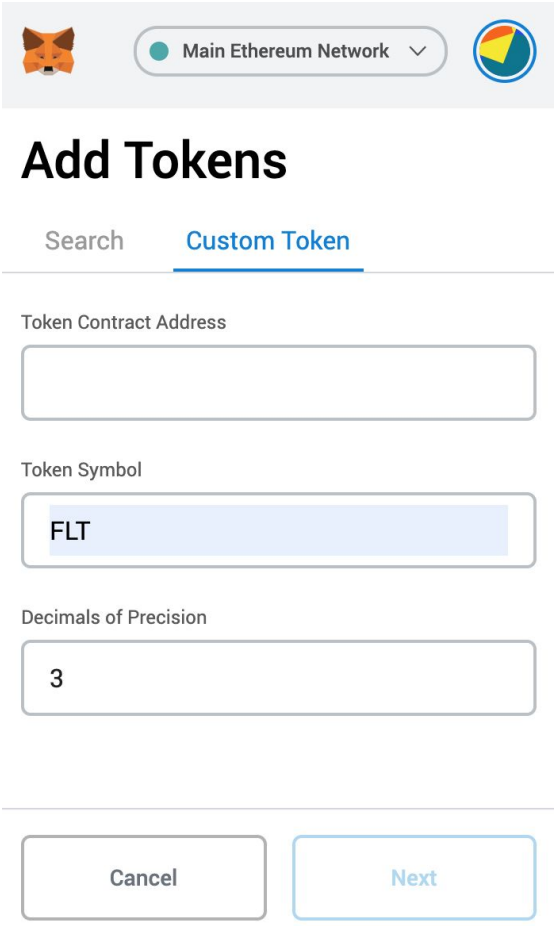
_buyerAddr: adres portfela kupującego bilet

_price (uint): cena za jeden bilet

_time (uint): czas dodany jako timestamp

updateType (string): możliwe wartości to setVehicle - zmiana ustawienia typu pojazdu, setEnabled - ustawienie zablokowania trasy, setAttribute - ustawienie innych atrybutów trasy.

Aby użytkownik widział w swoim portfelu tokeny FLT, musi dodać dodać definicję tokenu FLT. Poniżej przedstawiamy przykład ustawienia tokenu FLT w aplikacji Metamask, jest tutaj ważne aby znać adres kontraktu tokenu, jego symbol oraz liczbę miejsc dziesiętnych. **UWAGA**: Wymaga zainstalowania dodatku Metamask w przeglądarce lub posiadania innego webowego portfela Ethereum korzystającego z technologii web3.js.



Flotea SDK

Cały projekt Flotea Transport jest rozdzielony na kilka części, których należy użyć na własne potrzeby wdrożenia projektu. W skład projektu wchodzą następujące repozytoria:

- https://github.com/flotea-io/transport-eth-contracts - kody źródłowe inteligentnych kontraktów w elementami ułatwiającymi wdrożenie w sieci Blockchain

- https://github.com/flotea-io/flotea-ico - kody źródłowe i skompilowany projekt ICO oraz konsensus grupy obsługującej kontrakt. Zawiera aplikację internetową informującą o ofercie tokenów FLT formą ICO, formularzami do zakupu tokenów i zarządzania operatorem, głosowanie w konsensensusie operatorów nad przeznaczeniem budżetów formą DAO (Zdecentralizowana organizacja).
- https://github.com/flotea-io/transport-engine - Narzędzia SDK zawierają wszystko co agencja, sprzedaż, przewoźnik. Znajduje się tu również narzędzie Vagrant, które da się uruchomić na dowolnym systemie operacyjnym. SDK zawierają narzędzia oraz interfejs do wyszukiwania przejazdów, narzędzia do wytwarzania tras, listę zakupionych i sprzedanych biletów na trasie. Zawiera również API oraz bazę danych Postgres zsynchronizowaną z Blockchainem.

Flotea SDK Silnik Wyszukiwania

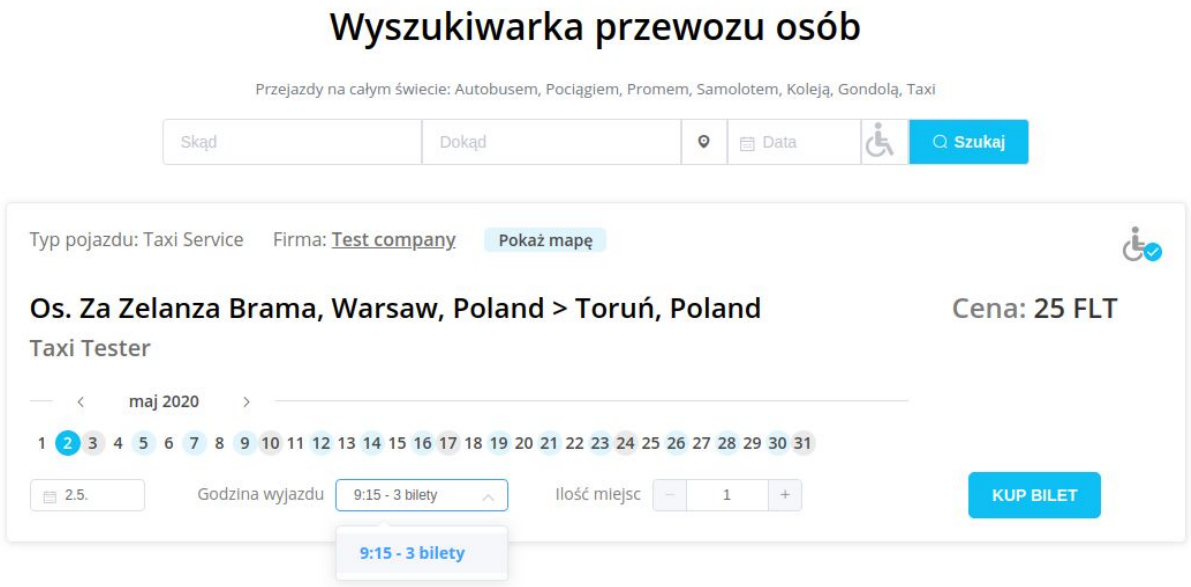
SKD Engine zawiera wytwarzanie tras przy pomocy różnych kryteriów, tak aby pasażer znalazł idealne połączenie, którego poszukuje. Lista kryteriów:

- ToLat, ToLng - lokalizacje przyjazdu we współrzędnych
- Radius - maksymalne oddalenie od wyjazdu/przyjazdu
- Date - pole [rok, miesiąc, dzień]
- Wheelchair bool - czy przejazd jest możliwy dla ludzi z niepełnosprawnościami

Danymi wyjściowymi jest lista tras i ich czasy wyjazdu i przyjazdu zdefiniowane w formacie JSON. Wyszukiwanie według czasu jest wykonane rozszerzeniem napisanym w języku GO w bazie danych Postgres, który parsuje informacje z JSON z każdej z tras i testuje czy pasuje do wyszukiwanej daty. Podłączony moduł został stworzony w GO dla szybkiego wykonywania. Sam kod

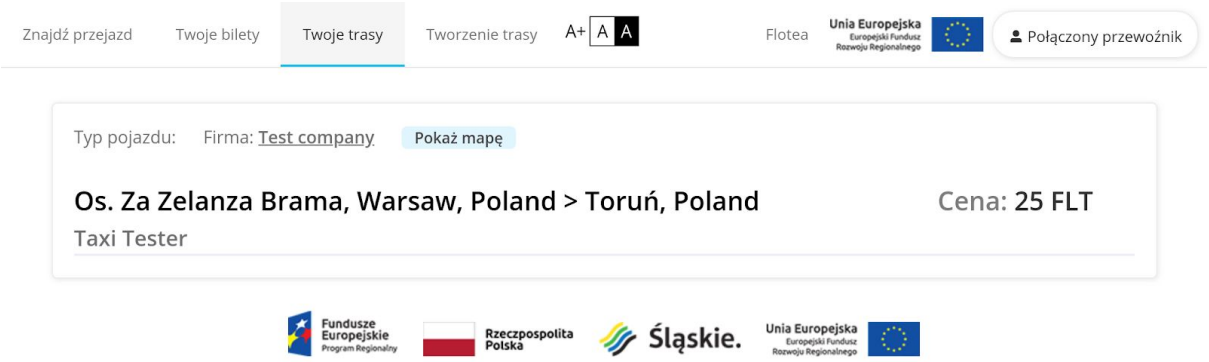
potrafi nie tylko kontrolować czy transport będzie wyjeżdżał w dany dzień, ale ma także możliwość podania zakresu czasowego, przy czym wyjściem będzie lista dat i czasów wyjazdu. Ten dodatek automatycznie się instaluje w przestrzeni testowej Vagrant, który jest umieszczony pod adresem /app/src/flt/utils pod nazwą pgtimespan.

Poniższy zrzut ekranu przedstawia wynik wyszukiwania z możliwością zakupu biletu. Widać na nim typ pojazdu, nazwę przewoźnika realizującego przejazd, opcję pokazania mapy, skąd dokąd przebiega trasa oraz mały kalendarz pokazujący wszystkie dni w tygodniu w których dostępna jest trasa (przy czym wszystko działa bardzo szybko, wyniki nawet w tysiącach są podawane w sekundę).



Przykład listy biletów pasażera:

Przykład listy tras przewoźnika:



Flotea SDK API

Po uruchomieniu środowiska testowego z Vagrant, Engine API wyszukiwarki, znajduje się pod adresem lokalnego serwera testowego: https://api.engine.devel

GET /v1/search

Wyszukuje i wraca informacje o znalezionych trasach

Przykład parametrów GET:

```
{
   "Radius":40000, // Odległość od pozycji na mapie
   "Date":[ // Data wyjazdu
      "2020", // Rok
      "05", // Miesiąc
      "04" // Dzień
   ],
   "FromLat": 50.052652, // Pozycja
   "FromLng": 19.987345, // Odjazd
   "ToLat": 52.237695, // Pozycja
   "ToLng": 21.005427, // Przyjazd
   "Wheelchair":0 // Obsługa osób z niepełnosprawnościami
}
```

Odpowiedź:


```
[
   {
      "RouteId":1,
      "RouteDesc":"Taxi Tester",
      "RouteType":114,
      "TripWallet":"0xe4516f7b9425338632a4674aeb793bec7d472054",
      "Places":3,
      "Schedule":"{\"and\": [{\"wd\": [2, 4, 6]}, {\"or\":
[{\"bev\": [2020, 4, 24], \"eev\": [2020, 8, 29, 23, 59, 59]}]},
{\"or\": [{\"hr\": [1], \"bhr\": [9, 15]}]}]}",
      "CurrencyType":"FLT",
      "Price":25,
      "FromLabel":"Os. Za Zelanza Brama, Warsaw, Poland",
      "ToLabel":"Toruń, Poland",
      "FromLat":52.237694,
      "FromLng":21.005426,
      "ToLat":53.027473,
      "ToLng":18.615425,
      "AgencyId":0,
      "AgencyName":"Test company",
      "AgencyUrl":"my.page"
   }
]
```


POST /v1/search/passager-tickets

Zwraca zakupione bilety na dany identyfikator portfela oraz detale trasy

Parametr: wallet (string): adres portela

Przykład:

```
{ wallet: "0x568a7332ce1236cf0a2433006eaa078d9c924cf7" }
```

Odpowiedź:


```
{
   "Tickets":[
      {
         "Count":1,
```

```
            "Time":1587806100,
            "RouteId":1
         }
      ],
      "Routes":[
         {
            "RouteId":1,
            "RouteDesc":"Taxi Tester",
            "RouteType":114,

"TripWallet":"0xe4516f7b9425338632a4674aeb793bec7d472054",
            "Places":3,
            "Schedule":"{\"and\": [{\"wd\": [2, 4, 6]}, {\"or\":
[{\"bev\": [2020, 4, 24], \"eev\": [2020, 8, 29, 23, 59, 59]}]},
{\"or\": [{\"hr\": [1], \"bhr\": [9, 15]}]}]}",
            "CurrencyType":"FLT",
            "Price":25,
            "FromLabel":"Os. Za Zelanza Brama, Warsaw, Poland",
            "ToLabel":"Toruń, Poland",
            "FromLat":52.237694,
            "FromLng":21.005426,
            "ToLat":53.027473,
            "ToLng":18.615425,
            "AgencyId":0,
            "AgencyName":"Test company",
            "AgencyUrl":"my.page",
            "Wheelchair":1
         }
      ]
}
```


POST /v1/search/buyed-tickets-in-times/ route_id

Zwraca liczbę zakupionych biletów na trasie w czasach

Parametry:

- route_id (int): identifikator trasy
- times (string): czasy oddzielone przecinkiem w formacie timestamp

Przykład:

```
{ times: "1588410900,1588670100,1588842900" }
```

Odpowiedź:

```
{ 1588410900: 2, 1588842900: 8 }
```

POST /v1/search/carrier-routes

Zwraca listę tras przewoźnika

Parametr: wallet (string): adres portfela

Przykład:

```
{ wallet: 0x26c68459ad0ab1298067b020dfef8fb0eaebc0ce }
```

Odpowiedź:


```
{
   "Tickets":{
      "1":{
         "1587806100":[
            {
               "TicketId":0,

"BuyerWallet":"0x568a7332ce1236cf0a2433006eaa078d9c924cf7"
            }
         ]
      }
   },
   "Routes":[
      {
         "RouteId":1,
         "RouteDesc":"Taxi Tester",
         "RouteType":114,
        "TripWallet":"0xe4516f7b9425338632a4674aeb793bec7d472054",
         "Places":3,
         "Schedule":"{\"and\": [{\"wd\": [2, 4, 6]}, {\"or\":
[{\"bev\": [2020, 4, 24], \"eev\": [2020, 8, 29, 23, 59, 59]}]},
{\"or\": [{\"hr\": [1], \"bhr\": [9, 15]}]}]}",
         "CurrencyType":"FLT",
         "Price":25,
         "FromLabel":"Os. Za Zelanza Brama, Warsaw, Poland",
         "ToLabel":"Toruń, Poland",
         "FromLat":52.237694,
```

```
            "FromLng":21.005426,
            "ToLat":53.027473,
            "ToLng":18.615425,
            "AgencyId":0,
            "AgencyName":"Test company",
            "AgencyUrl":"my.page"
        }
    ]
}
```

Oprócz klasycznego API, każdy z kontraktów posiada publiczny interfejs, który jest dostępny w zdecentralizowanej bazie danych Blockchain. Każdy z tych interfejsów został opisany w tym dokumencie.

Flotea SDK - Sposób synchronizacji z Blockchain

Elementem SDK jest moduł do synchronizowania kontraktu na Blockchainie z lokalną bazą danych w przestrzeni serwera (np. Testowego Vagrant). Synchronizacja przebiega formą rejestrowania się do zdarzeń zdefiniowanych w kontraktach na Blockchainie, gdzie definiuje się adres kontraktu, w tym przypadku adresu kontraktu Transport. Zrzut ekranu poniżej przedstawia część kodu, przy pomocy którego zmienna query filtruje informacje tylko z adresu kontraktu Transport. Inicjalizujemy zmienną logs, do której będą zapisywane przefiltrowane elementy. Metodą *SubscribeFilterLogs* odbywa się rejestracja odbioru wiadomości z blockchain.

```
query := ethereum.FilterQuery{
    Addresses: []common.Address{gTransportAddress},
}

logs := make(chan types.Log)

sub, err := gClient.SubscribeFilterLogs(context.Background(), query, logs)
if err != nil {
    fmt.Println(err)
}
```

Dalej następuje dzielenie odebranych wiadomości zgodnie z tematem, tj. według nazwy zdarzenia, a następnie uzyskane wartości zapisujemy w lokalnej bazie danych w standardzie GTFS.

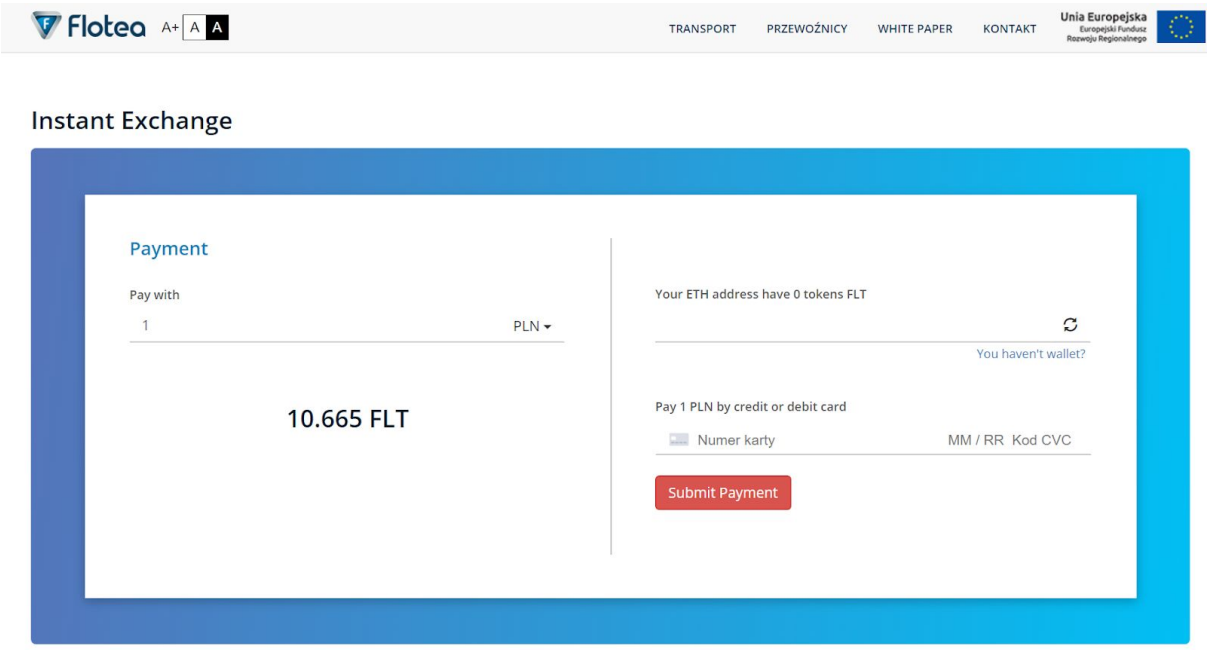
Flotea SDK - System płatności

W systemie istnieje oprogramowanie pozwalające na zakup tokenów FLT za pomocą kryptowaluty Ethereum jak i złotówek. Mechanizm ma formę ICO podzieloną na etapy. Każda z faz posiada ilość tokenów do sprzedaży w określonej cenie, a im wcześniej zostanie wykonany zakup - tym cena jest niższa. ICO jest przemyślane w ten sposób, aby zapewnić mechanizm kupowania Tokenów przez podróżnych, przewoźników oraz inwestorów. Nie ma ograniczenia czasowego dla całego mechanizmu. Jest natomiast ograniczenie ilościowe. W kontrakcie istnieje założenie, że wyemitowane zostanie 72 000 000 tokenów FLT w formie sprzedaży, przy założeniu, że w ostatnim etapie jeden token kosztuje 0.09 zł. Oczywiście jest to mechanizm przygotowany do działania, nie został uruchomiony w sieci publicznej Blockchain.

		CENA ZA TOKEN W	
ETAPY	TOKENÓW	PLN	ETH
1	5 000 000	0.01	0.00001047
2	3 000 000	0.02	0.00002094
3	1 000 000	0.03	0.00003141
4	1 000 000	0.04	0.00004188
5	1 000 000	0.05	0.00005235
6	1 000 000	0.06	0.00006282
7	60 000 000	0.09	0.0001047

Poniższy zrzut ekranu przedstawia formularz zakupu Tokenów w formie wymiany kryptowaluty ETH lub zakupu za złotówki. Płatność w złotówkach jest obsługiwana przez integrację z systemem płatności on-line Stripe[2], który umożliwia wygodny i bezpieczny zakup kartą kredytową. Natomiast płatność w ETH odbywa się za pomocą przelewu poprzez dodatek do przeglądarki Metamask. Może być również wykonany manualnie przelewem z dowolnego portfela, czy nawet w przypadkach zaawansowanego użytkownika z linii poleceń (możliwe jest w ten sposób zintegrowanie zakupu w innych rozwiązaniach web).

[2] https://stripe.com/en-pl

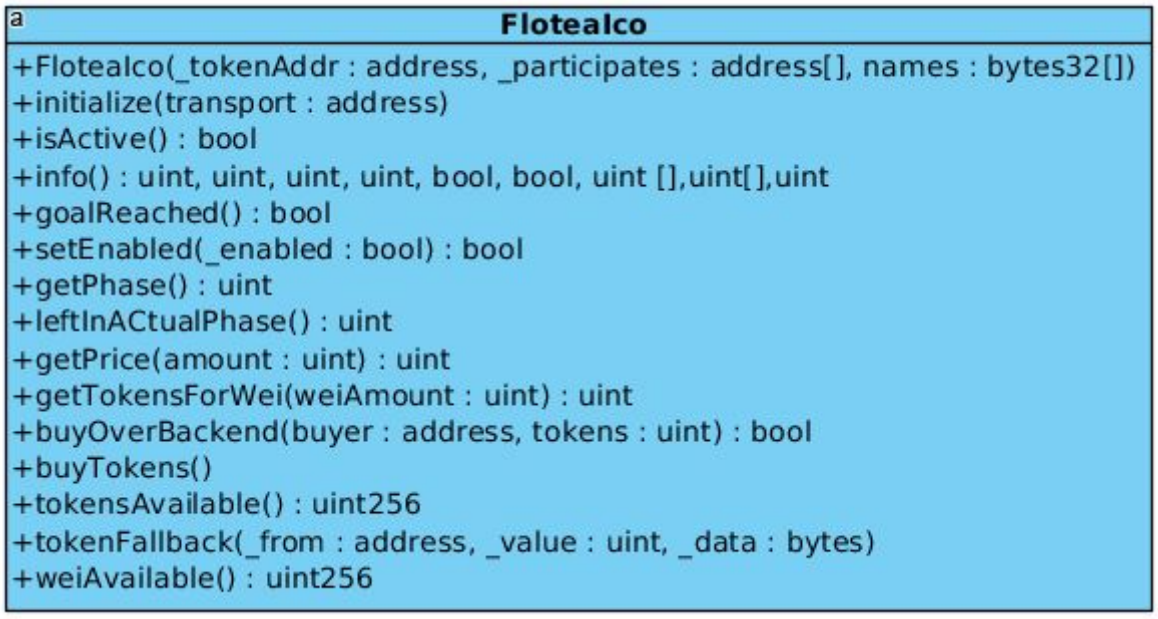


System sprzedaży tokenu FLT zapewnia kontrakt FlotealCo i backendowa aplikacja napisana w języku GO. Konstruktor kontraktu zawiera parametry *_tokenAddress* (adres tokenu FLT), adresy *_participates* oraz imiona właścicieli (konsensusu). Aby było możliwe przesunięcie tokenów sprzedażą biletów, jest ważne aby ten kontrakt zainicjować z parametrem adresu kontraktu Transport. Wywołaniem funkcji kontraktu możemy uzyskać różne informacje, na przykład isActive() zwraca bool zmiennej true dopóki kontrakt jest zainicjalizowany, dopuszczony i nie było zrealizowanego celu (sprzedaż 72000000 tokenów FLT. Funkcja info() zwraca informację o stanie ICO:

- initialTokens *uint* - liczba tokenów FLT po inicjalizacji,
- tokensAvailable uint - aktualna liczba tokenów FLT na sprzedaż,
- phase uint - aktualna faza sprzedaży,
- leftInActualPhase uint - liczba tokenów, które jeszcze zostały w aktualnej fazie,
- initialized bool - zwraca czy kontrakt został zainicjalizowany,
- enabled bool - zwraca czy kontrakt posiada aktywną sprzedaż tokenów,
- prices uint[] - pole cen na fazę,
- tokens uint[] - pole ilości tokenów na fazy,

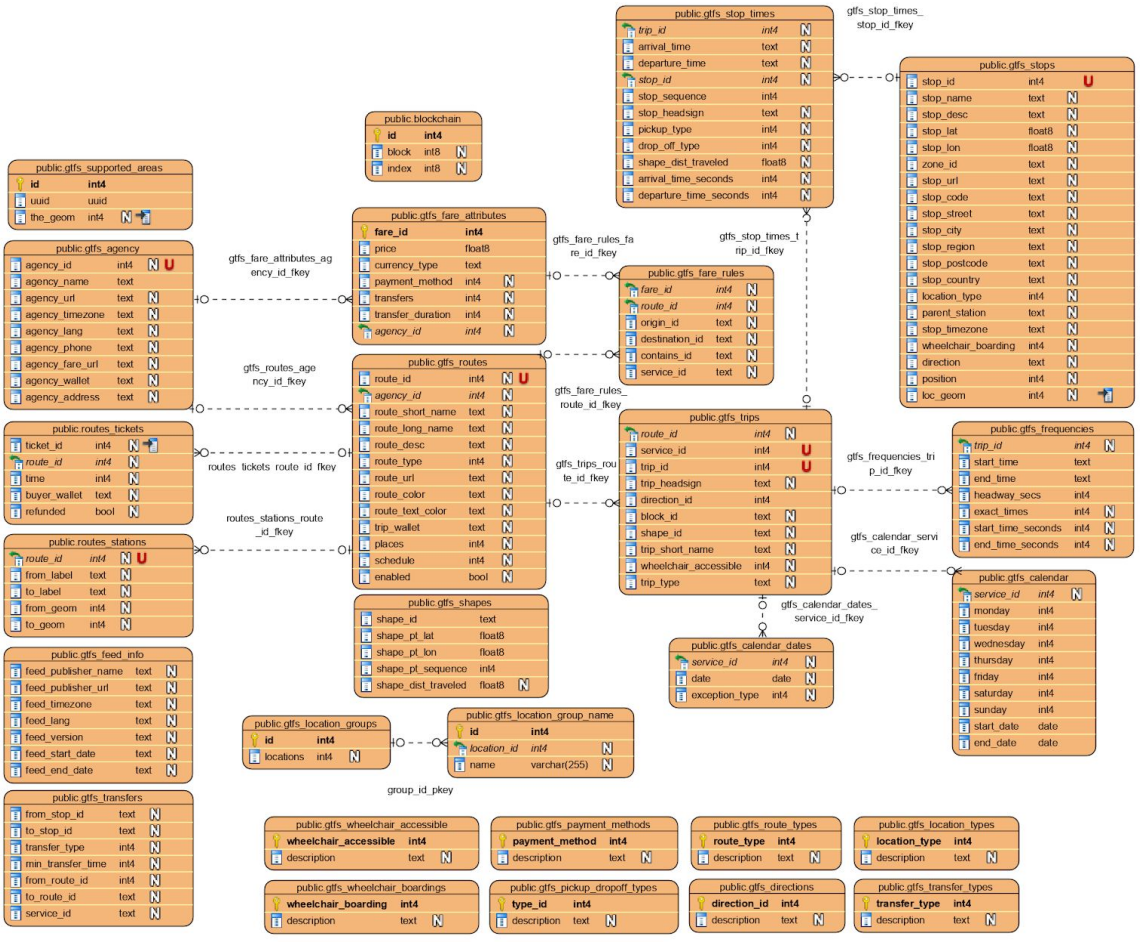
- balance uint - zawartość w kryptowalucie ETH

Sprzedaż tokenów da się zatrzymać, lub aktywować za pomocą funkcji *setEnabled()*. Funkcja getPrice() wylicza cenę w wartościach mierzonych w Wei na podstawową wymaganą liczbę tokenów i to w przypadku zmiany fazy. Dalsze funkcje obliczeniowe takie jak *getTokensForWei()* zwracają liczbę tokenów, które uzyskamy za podaną liczbę Wei. Do zakupu tokenów wystarczy wysłać na adres tego kontraktu ethereum przez specjalną funkcję interfejsu typu payable wywołać funkcję *buyTokens()*, którą da się także wywołać automatycznie, ta wyliczy ile należy przesunąć tokenów FLT na portfel, z którego była wykonywana płatność. Funkcją *buyOverBackend()* można wywołać jedynie właściciel kontraktu, parametry to adres kupującego - adres na który przesyłane są tokeny, uint tokens - ilość tokenów, które są do przesłania. Aby ta funkcja była wywołana potrzebne jest prawo - właściciela kontraktu, który posiada prywatne klucze. W naszym przypadku jest to klucz zapisany i ułożony w aplikacji backendowej napisanej w GO.



Baza danych projektu - diagram ERD

Baza danych została przygotowana w technologii PostgreSQL z dodatkiem PostGIS, migracje dostępne są w plikach migracyjnych projektu SDK i silnika, które uruchamiają się podczas inicjalizacji projektu. Baza danych została opracowana na podstawie analiz zamieszczonych w rozdziale dodatkowych analiz, zaleca się zapoznanie z analizami w celu łatwiejszego przyswojenia wiedzy na temat technologii GTFS.



Powyższy zrzut ekranu przedstawia diagram bazy danych projektu. Jest to ogólna struktura bazy danych powstała w oparciu o standard GTFS z modyfikacjami. Ze znaczących modyfikacji to dodano tabelę synchronizacji z Blockchain, która

przedstawia coś podobnego do migracji danych, aktualny stan synchronizacji z Blockchain. Kolejną znaczącą zmianą jest użycie własnego harmonogramowania tras, zastępując całkowicie tabele harmonogramu GTFS, przy czym tabele zostały zachowane dla przypadku kiedy żeby zapisać trasy w formacie GTFS lub odczytać trasy z formatu GTFS oraz, żeby łatwiej było wdrażać projekty, które już ten standard obsługują.

Kluczowe rozwinięcia GTFS:

a) Tabela gtfs_agency - dodano kolumnę na adres portfela agencji, adres agencji, url - stronę internetową,
b) Tabela gtfs_routes - dodano kolumnę adresu portfela trasy, ilość miejsc, kolumnę na zapisany harmonogram w formacie JSOB, informację czy trasa jest dostępna.
c) Stworzono dodatkową tabelę routes_station w oparciu o postGIS i typ danych geometry, które są punktami adresu wyjazdu i adresu docelowego. W celach użycia wydajniejszego przeszukiwania w oparciu o dane lokalizacji.
d) Stworzono tabelę routes_tickets - która jest powiązana z tabelą routes. Posiada dane takie jak moment zakupu, portfel kupującego, czy płatność została zwrócona itp.

III. Przyjęte założenia

Założenia technologiczne

Lp.	Obszar	Założenie technologiczne
1.	Infrastruktura	Zamieszczenie projektu w dowolnej infrastrukturze, szybkie uruchomienie środowiska testowego (instrukcje Vagrant). System Linux. Możliwe do uruchomienia na serwerach prywatnych, publicznych, VPS. Możliwość integracji z istniejącymi infrastrukturami portali.

2.	Sieć i bezpieczeństwo	Istotnym elementem projektu jest sieć blockchain, zapewniająca bezpieczeństwo integracji danych oraz zdecentralizowany sposób i dostęp do danych nt. Tras. Inteligentne Kontrakty mogą istnieć w sieci Blockchain oraz dostarczać
3.	Standardy wymiany danych	Standardy API REST. Publiczne Interfejsy Inteligentnych Kontraktów udostępnione bezpośrednio w sieci Blockchain. Harmonogramy wyjazdów w oparciu o JSON, wydajny standard danych w kompresji i zapisie do Inteligentnego Kontraktu.
4.	Systemy operacyjne serwerów	Preferowany Linux, projekt wytwarzany w systemie Ubuntu 18.04, przetestowany w Ubuntu 20.04
5.	Bazy danych	PostgreSQL w wersji 11 z uwagi na wspieranie możliwości pisania rozszerzeń w języku go (obecnie da się jedynie bez obsługi LLVM). Jest to baza danych open source, która posiada bardzo wydajne rozwiązania ułatwiające prace na danych geograficznych, które zostały użyte w projekcie PostGIS. Uwaga: rozwiązania nie udało się uruchomić z PostgreSQL w wersji 12 i PostGIS 3.0 z uwagi na LLVM. W tym wypadku rozszerzenie albo - należy przygotować we własnym zakresie, albo wesprzeć w rozwoju rozszerzenie https://gitlab.com/microo8/plgo, do wspierania PostgreSQL 11.
6.	Portale	Użyto technologii GO z zastosowaniem wzorca architektury MVC z widocznym rozdzieleniem danych (frontend, backend). Technologię Frontend wykonano z użyciem frameworka Vue.js z dodatkiem Nuxt.

Założenia licencyjne

Oprogramowanie udostępniono na licencji GNU General Public License v3.0. Powszechna Licencja Publiczna GNU jest wolną, opartą na zasadzie “copyleft”, licencją na oprogramowanie i na innego rodzaju utwory. Twórcy oprogramowania korzystający z Powszechnej Licencji Publicznej GNU chronią swoje prawa dwuetapowo: (1) zastrzegają prawa autorskie do oprogramowania oraz (2) oferują niniejszą Licencję, udzielając prawnego zezwolenia na kopiowanie, rozpowszechnianie i/lub modyfikację tego oprogramowania. Licencje zostały umieszczone w kodzie źródłowym jako pliki LICENSE i zawierają angielską wiążącą treść licencji.

IV. Instrukcja wdrożenia

Vagrant

Instrukcje wdrożenia - część przykładowych instrukcji wdrożenia zostały zbudowane za pomocą plików sh, poleceń, które pozwalają zbudować testowe środowisko, w którym od razu można rozpocząć pracę. Bazą pod instrukcje jest plik Vagrantfile umieszczony bezpośrednio w projekcie. W nim znajdują się odnośniki do plików instrukcji komend, które pozwalają zbudować testowe środowisko lub mogą być przydatne w przypadku docelowego wdrożenia. Pliki są umieszczone w katalogu Vagrant / provision i są one podzielone wg. Sposobu uruchomienia (jednokrotnie z prawami root, za każdym razem z prawami root, za każdym razem z prawami użytkownika systemu). Wykonanie komend jest automatyczne w przypadku użycia systemu Vagrant (za pomocą komendy vagrant up). Jednakże, w przypadku chęci przygotowania lub dostosowania systemu operacyjnego manualnie, należy zbudować środowisko zgodne z instrukcjami. Uruchomienie automatyczne - powinno cały projekt uruchomić automatycznie, wykonując polecenia krok po kroku, jednakże należy zwrócić uwagę, że czasem są konieczne modyfikacje manualne np. z powodów aktualizacji elementów zewnętrznych.

Pliki instrukcji znajdują się odpowiednio:

a) Dla jednorazowego uruchomienia z prawami root
   src/vagrant/provision/always-as-root.sh
b) Dla wielokrotnego uruchomienia z prawami root
   src/vagrant/provision/once-as-root.sh
c) Dla wielokrotnego uruchomienia z prawami user
   src/vagrant/provision/always-as-vagrant.sh

Konfiguracje dla Engine SDK

Instalacja w systemie curl, ca-certificates, apache2, phppgadmin (do zarządzania bazą danych) oraz paczki build-essential.

```
info "Install additional software"
sudo DEBIAN_FRONTEND=noninteractive apt-get install -y git curl
ca-certificates apache2 --assume-yes phppgadmin --assume-yes
build-essential
echo "Done!"
```

Instalacja baz danych PostgreSQL w wersji 11 z dodatkiem PostGIS 2.5 oraz narzędzi deweloperskich dla postgres 11 służących do ewentualnej kompilacji dodatku do bazy danych (np. W przypadku konieczności wprowadzenia stosownych poprawek dla dodatku).

```
info "Install PostgreSQL"
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt
disco-pgdg main" >> /etc/apt/sources.list'
wget --quiet -O -
http://apt.postgresql.org/pub/repos/apt/ACCC4CF8.asc | sudo
apt-key add -
apt-get update
apt-get install -y postgresql-11-postgis-2.5
apt-get install -y postgresql-server-dev-11
echo "Done!"
```

Podstawowe konfigurowanie dodatku PostgreSQL - zaakceptowanie wszystkich połączeń na IPv4 - tylko dla rozwoju aplikacji, rozbudowy lub w celach testowych. W przypadku produkcji należy zadbać o prawidłową konfigurację zabezpieczeń bazy danych PostgreSQL.

```
info "Configure PostgreSQL database"
# fix permissions
sed -i "s/#listen_address.*/listen_addresses = '*'/"
/etc/postgresql/11/main/postgresql.conf
#fixing postgres pg_hba.conf file
```

```
cat >> /etc/postgresql/11/main/pg_hba.conf <<EOF
#Accept all IPv4 connections - FOR DEVELOPMENT ONLY!!!
host    all         all         0.0.0.0/0             md5
EOF
echo "Done!"
```

Wytworzenie użytkownika, hasła, roli i baz dla platformy.

```
info "PostgreSQL: Create User and Database for dev and tests"
# create user and database
su postgres -c "psql -c \"CREATE ROLE flotea SUPERUSER LOGIN
PASSWORD 'HASŁO!!!'\" "
su postgres -c "createdb -E UTF8 -T template0 --locale=en_US.utf8
-O flotea flotea_platform"
#create database for tests
su postgres -c "createdb -E UTF8 -T template0 --locale=en_US.utf8
-O flotea flotea_platform_test"
echo "Done!"
```

Instalacja dodatku *pgtimespan* wytworzonego w trakcie projektu.

```
info "Install PostGIS pgtimespan Extension"

/bin/mkdir -p '/usr/share/postgresql/11/extension'
/bin/mkdir -p '/usr/share/postgresql/11/extension'
/bin/mkdir -p '/usr/lib/postgresql/11/lib'
/usr/bin/install -c -m 644
/app/src/flt/utils/pgtimespan/build/pgtimespan.control
'/usr/share/postgresql/11/extension/'
/usr/bin/install -c -m 644
/app/src/flt/utils/pgtimespan/build/pgtimespan--0.1.sql
'/usr/share/postgresql/11/extension/'
/usr/bin/install -c -m 755
/app/src/flt/utils/pgtimespan/build/pgtimespan.so
'/usr/lib/postgresql/11/lib/'

su postgres -c 'psql -d flotea_platform -c "CREATE EXTENSION
pgtimespan;"'
```

W osobliwym przypadku kiedy konieczne jest przekompilowanie dodatku np. w przypadku innej wersji bazy postgreSQL, albo chęci dokonania zmian w algorytmie wyszukiwania na potrzeby własnego wdrożenia. Instrukcja wygląda następująco:

1. Należy zainstalować dodatek plgo (środowisko go lang)
2. plgo musi być dodany w zmiennych środowiskowych /etc/environment PATH
3. Po zmianach kodu należy uruchomić plgo ./ w folderze pgtimespan, który wygeneruje BUILD
4. Wejść w katalog BUILD i wykonać polecenie make install with_llvm=no (bez llvm).
5. Od tego momentu plugin powinien być zainstalowany
6. Z poziomu bazy danych należy wykonać instrukcję CREATE EXTENSION pgtimespan;

Zbudowanie, przetestowanie zmian i aktualizacja dodatku odbywa się w następujący sposób:

```
/app/src/flt/utils/pgtimespan# plgo
/app/src/flt/utils/pgtimespan# cd build
/app/src/flt/utils/pgtimespan/build# make install with_llvm=no
/app/src/flt/utils/pgtimespan/build# service postgresql restart

DROP EXTENSION pgtimespan; // if exists before
CREATE EXTENSION pgtimespan; // updated
```

Przykład użycia:

```
SELECT * FROM gtfs_routes WHERE IsOpen(schedule::TEXT,
ARRAY[2019,10,12]);
```

Powyższe zapytanie, sprawdzi harmonogramy wyjazdów zapisane w kolumnie schedule tablicy gtfs_routes, czy danych wyjazd spełnia wymagania dnia 2019.10.12 w przypadku pozytywnym zwraca wyniki tj. Wybrane trasy, których harmonogramy pozwalają na wyjazd w danym czasie.

Do bazy PostgreSQL wymagane są następujące dodatki: postgis, uuid-oosp, pgcrypto oraz przygotowany w tym projekcie dodatek pgtimespan. Dodatki mogą być dodane w momencie migracji danych - jest to pierwszy plik migracji bazy danych w projekcie, który znajduje się w pliku *20190414_094510_enable_extensions.go* znajdującym się w katalogu *src/flt/database/migrations* projektu.

Konfiguracje dla Środowiska testowego Apache

Podstawowa domena engine.devel (dla domeny testowej engine.devel)

```
<VirtualHost *:80>
        ServerName engine.devel
        Redirect permanent / https://engine.devel/
</VirtualHost>
<VirtualHost *:443>
    ServerName engine.devel
    DocumentRoot /app/src/frontend/dist
    SSLEngine on
    SSLCertificateFile /home/vagrant/engine.devel.cert
    SSLCertificateKeyFile /home/vagrant/engine.devel.key
    DirectoryIndex index.html
    <Directory /app/src/frontend/dist>
        Require all granted
    </directory>
</VirtualHost>
```

Konfiguracja domeny api.engine.devel dla środowiska testowego

```
<VirtualHost *:80>
        ServerName api.engine.devel
        Redirect permanent / https://api.engine.devel/
</VirtualHost>
<VirtualHost *:443>
    ServerName api.engine.devel

    SSLEngine on
    SSLCertificateFile /home/vagrant/engine.devel.cert
    SSLCertificateKeyFile /home/vagrant/engine.devel.key

    ProxyRequests Off
```

```
    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>
    ProxyPass / http://192.168.66.66:8010/
    ProxyPassReverse / http://192.168.66.66:8010/
</VirtualHost>
```

Uwaga: Adresy IP oraz ewentualne certyfikaty należy zastąpić własnymi ustawieniami, chyba, że jest to środowisko stawiane z pliku vagrant w celach testowych.

Konfiguracja domeny wss.engine.devel dla połączenia WEB SOCKET

```
<VirtualHost *:443>
    ServerName wss.engine.devel

    SSLEngine on
    SSLCertificateFile /home/vagrant/engine.devel.cert
    SSLCertificateKeyFile /home/vagrant/engine.devel.key

    ProxyRequests Off
    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>
    ProxyPass / ws://192.168.66.66:8010/
    ProxyPassReverse / ws://192.168.66.66:8010/
</VirtualHost>
```

Konfiguracja domeny frontend.engine.devel

```
<VirtualHost *:80>
        ServerName frontend.engine.devel
        Redirect permanent / https://frontend.engine.devel/
</VirtualHost>
<VirtualHost *:443>
    ServerName frontend.engine.devel

    SSLEngine on
    SSLCertificateFile /home/vagrant/engine.devel.cert
    SSLCertificateKeyFile /home/vagrant/engine.devel.key

    ProxyRequests Off
    <Proxy *>
        Order deny,allow
```

```
        Allow from all
    </Proxy>
    ProxyPass / http://192.168.66.66:3000/
    ProxyPassReverse / http://192.168.66.66:3000/
</VirtualHost>
```

Instalacja struktury bazy danych

W projekcie wykonano pliki migracji danych, które należy uruchomić w celach zbudowania struktury bazy danych. Pliki migracji znajdują się w katalogu *src/flt/database/migrations.* Jest to struktura zbudowana na bazie standardu GTFS z modyfikacjami potrzebnymi do działania projektu (np. harmonogramu). Aby uruchomić migrację należy posiadać poprawnie zainstalowany język golang, dodatek do go framework beego.

```
chmod 777 /app/src/flt/migrate
. /app/src/flt/migrate
```

Który jest skrótem od polecenia:

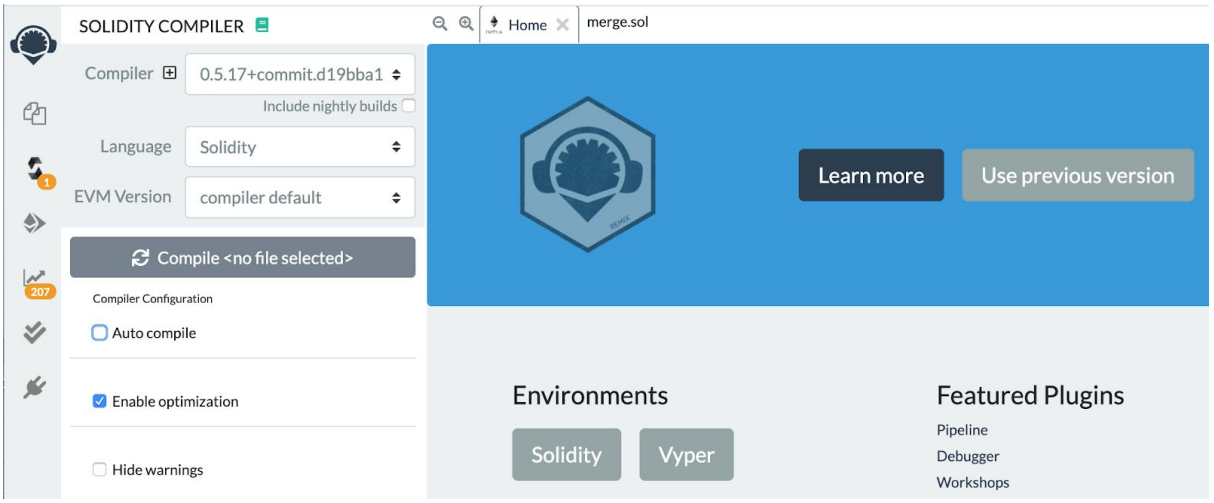
```
bee migrate -driver=postgres
-conn="postgres://flotea:Hasło!@127.0.0.1:5432/flotea_platform?ssl
mode=disable"
```

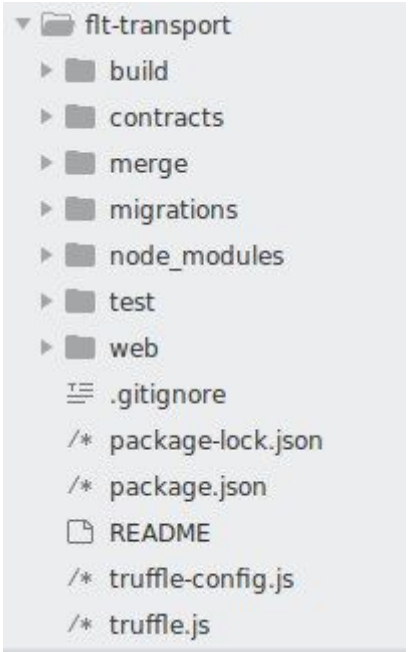
Uwaga: Struktura bazy danych zainstaluje się automatycznie w przypadku użycia Vagrant.

Wdrożenie kontraktów do sieci Ethereum

W katalogu merge jest już wytworzony plik ze wszystkimi kontraktami, ten nagrywamy do internetowego narzędzia Remix - Ethereum IDE oraz należy wybrać środowisko Solidity. Następnie należy pozwolić na optymalizację oraz skompilować kontrakty. Poniższy widok przedstawia aktualny interfejs narzędzia webowego

Solidity Compiler. W zaawansowany sposób, można użyć bezpośrednio narzędzi linii poleceń na lokalnym komputerze podłączonym do sieci (kompilator solc i parametry). Jednakże w celu uproszczenia kompilacji zaleca się korzystanie z Remix.





Zrzut ekranu widoczny po lewej stronie - to struktura folderu kontraktów. Wszystkie kontrakty znajdują się w folderze contracts, natomiast w folderze merge znajduje się jeden plik, który stanowi połączone w jeden plik rozwiązanie, które można w łatwy sposób użyć w aplikacji Remix[3] Solidity Compiler.

Ponieważ kontrakty mają między sobą powiązania i wewnętrzne kontrole bezpieczeństwa, należy wdrożyć kontrakty zgodnie z następującymi krokami:

1. FloteaToken
2. VotingCarrier - potrzebne są dwa adresy, które na początku będą decydowały o przyjmowaniu dalszych adresów / portfeli do konsensusu przewoźników. (Jeżeli w tym miejscu doda się więcej adresów

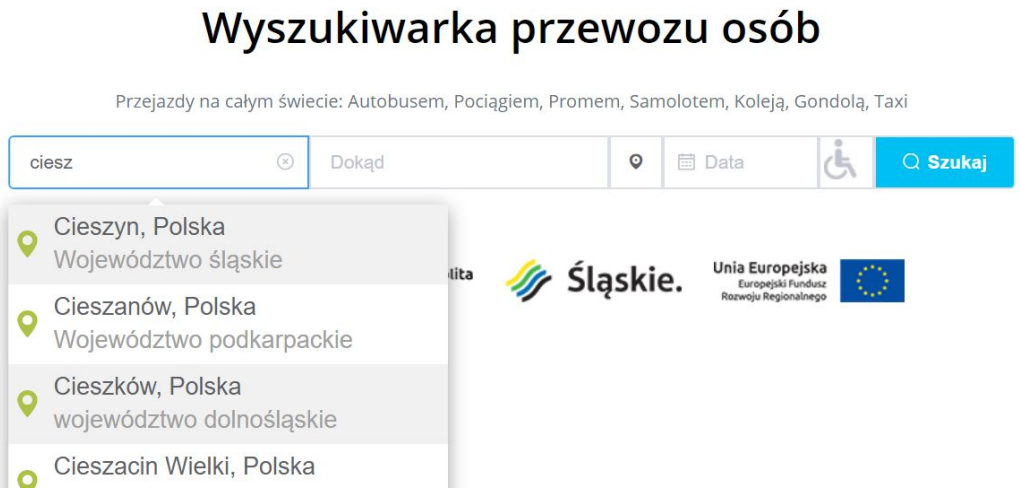
[3] https://remix.ethereum.org

niż dwa, można w późniejszym czasie na zasadzie głosowania i funkcjonalności kontraktu - konta te usunąć).

3. Carriers - wymaga utworzonego kontraktu VotingCarrier
4. FlotealCo - Pierwszy parametr przy wytwarzaniu to adres tokenu FLT, drugi to lista trzech adresów tworzących konsensus, natomiast ostatni parametr jest listą trzech nazw, które dotyczą drugiego parametru.
5. Transport - ostatni kontrakt wymaga adresów wszystkich wytworzonych wcześniej kontraktów.

Opcjonalnie: Pelias

Istotnym elementem są podpowiedzi wyszukiwarki, w aplikacji stanowiącej przykładowy interfejs podpowiedzi te zostały zaimplementowane w wyszukiwarce projektu. Element ten mogą stanowić również inne wdrożenia np. Podpowiedzi google. W projekcie zastosowano jednak rozwiązanie polegające na otwartoźródłowym rozwiązaniu. Na stronie https://pelias.io oraz na stronie repozytorium https://github.com/pelias/pelias, dostępny jest projekt oraz instrukcje w różnych wariantach. Istnieje również środowisko do szybkiego wdrożenia w oparciu o technologię Docker, jednakże ograniczone.



Na zrzucie ekranu pokazany jest element podpowiedzi miejscowości, gotowy kod elementu prezentujący wyniki jest dostępny bezpośrednio w kodzie oprogramowania udostępnionego wraz z projektem. Polega on na wykorzystaniu wdrożenia pelias pod adresem https://geo.flotea.pl, który również można wykorzystać podczas wdrożeń / integracji w innych miejscach.

Technologia PELIAS.IO jest to otwartoźródłowy system z dostepnymi narzędziami służącymi do szeroko rozumianej pracy z danymi geograficznymi, funkcjami geocoding-u i reverse geocoding-u, odnajdywania miejsc, położeń geograficznych punktów itp. Technologia ta dostarcza całe API oraz bazuje na otwartoźródłowych danych z baz takich jak Geonames[4], WhosOnFirst[5]. Umożliwia także implementacje całych map OpenStreetMaps[6] oraz OpenAddress[7]. Dane z tych baz danych są importowane oraz budowana jest z nich lokalna z punktu widzenia projektu baza danych oraz dostępne narzędzia API służące do wydajnych wyszukiwań. Baza ta jest importowana do lokalnej bazy w technologii ElasticSearch. Rozwiązanie stanowi niezależny zasób danych geolokalizacyjnych i mapy dla lokalnych rozwiązań oraz ułatwione wyszukiwanie, parsowanie adresów i miejsc.

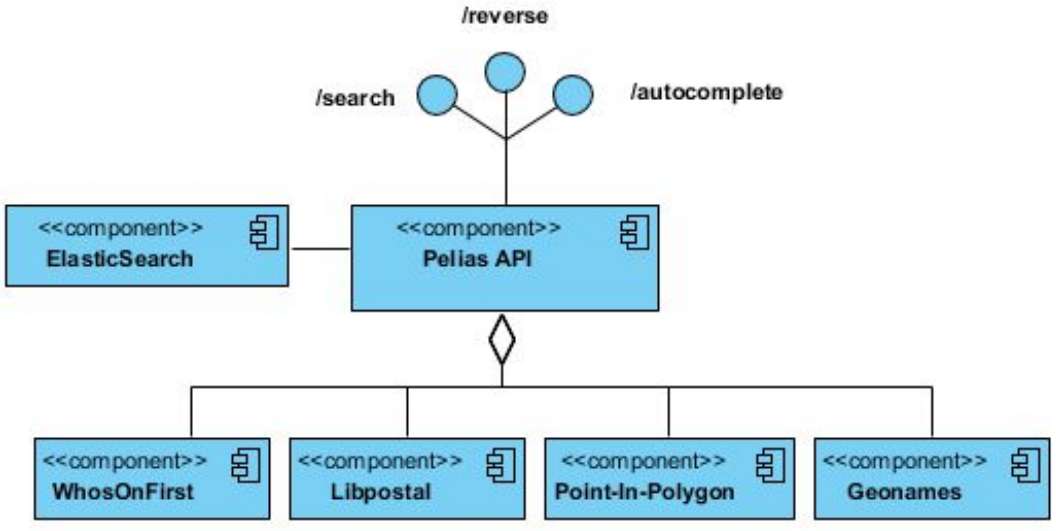
Instrukcja wdrożenia:

1. Konfiguracja Apache2 do obsługi serwisów, do przekierowania serwisów przez bezpieczne protokołu HTTPS / SSL.
2. miejsce-instalacji/api - Serwis API Pelias dostępny pod domeną geo.flotea.pl
3. miejsce-instalacji/geonames - Baza danych Geonames zaimportowana do ElasticSearch na bazie której działa API - tutaj wybrano obszar Europy
4. miejsce-instalacji/go-whosonfirst-libpostal - aplikacja, która łączy libpostal z API - tutaj wybrano obszar Europy
5. miejsce-instalacji/libpostal - aplikacja w C, która potrafi parsować adresy np. /search text adres ulica 43-400 cieszyn, ładnie przygotowuje json i wykryje ulicę, postal i miasto.

[4] https://www.geonames.org
[5] https://www.whosonfirst.org
[6] https://www.openstreetmap.org
[7] https://openaddresses.io

6. miejsce-instalacji/pip-service - serwis, który wpuszcza point na geometry i wybiera z niego regiony administracyjne, działa w /search
7. miejsce-instalacji/placeholder - serwis, który potrafi z niedokończonych zdań wyłuskać wyniki np. Jastrz zdr, zwraca “jastrzębie-zdrój”
8. miejsce-instalacji/schema - aplikacja schema
9. miejsce-instalacji/services - tutaj umieszczone zostały pliki .service odpowiadające serwisom dostępnym w systemie (ponieważ niektóre usługi uruchomione zostały jako serwis. Chodzi o: ***pelias.api, pelias.libpostal, pelias.pip, pelias.placeholder***
   Serwisy te jako root możemy resetować i uruchamiać np. ***service pelias.api status***
10. miejsce-instalacji/whosonfirst - źródło danych dla API z WhosOnFirst
11. Plik **pelias.json** dostępny z poziomu miejsce-instalacji/pelias.json jest plikiem konfiguracyjnym usługi API (i innych usług z pelias)

Struktura wdrożenia pelias w geo.flotea.pl:



Przykłady użycia:

1. ***Autocomplete*** - polecenie służące do szybkiego zdobywania podpowiedzi, potrafi obsłużyć (w zależności od serwera) do kilkuset poleceń na sekundę (a nawet do tysięcy w przypadku, kiedy serwer jest bardzo mocny). Wyniki można precyzować używając odpowiednich parametrów w zapytaniu spowoduje, że widoczne będą jedynie elementy związane z danymi "warstwami", można wtedy ograniczyć pokazywanie wyników, które nas nie interesują. (więcej w dokumentacji pelias)

```
?text=poland&layers=locality,country,macroregion,region,localadmin,count
ry
```

Polecenie:
https://geo.flotea.pl/v1/autocomplete?text=cieszyn

Wynik:


```
{
 "geocoding": {
    "version": "0.2",
    "attribution": "http://geo.flotea.pl/attribution",

"query": {
      "text": "cieszyn",
      "parser": "addressit",
      "parsed_text": {},
      "tokens": [
        "cieszyn"
      ],
      "size": 10,
      "private": false,
      "lang": {
        "name": "Polish",
        "iso6391": "pl",
        "iso6393": "pol",
        "defaulted": false
      }
    },
    "engine": {
      "name": "Pelias",
```

```
      "author": "Mapzen",
      "version": "1.0"
    },
    "timestamp": 1553678652357
  },
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          18.79147,
          49.72628
        ]
      },
      "properties": {
        "id": "7530808",
        "gid": "geonames:county:7530808",
        "layer": "county",
        "source": "geonames",
        "source_id": "7530808",
        "name": "Powiat cieszyński",
        "accuracy": "centroid",
        "country": "Polska",
        "country_gid": "whosonfirst:country:85633723",
        "country_a": "POL",
        "region": "Województwo śląskie",
        "region_gid": "whosonfirst:region:85687277",
        "region_a": "SL",
        "county": "Cieszyński",
        "county_gid": "whosonfirst:county:102079421",
        "continent": "Europa",
        "continent_gid": "whosonfirst:continent:102191581",
        "label": "Powiat cieszyński, Polska"
      }
    },
  ],
  "bbox": [
    15.71673,
    49.72628,
    21.60572,
    54.17065
  ]
}
```


2. ***Reverse*** - polecenie zwracające wynik, miejsce które znajdują się pod danym punktem geograficznym.

Polecenie:

https://geo.flotea.pl/v1/reverse?point.lon=18.63213&point.lat=49.75133

Wynik:


```
{
  "geocoding": {
    "version": "0.2",
    "attribution": "http://geo.flotea.pl/attribution",
    "query": {
      "size": 10,
      "private": false,
      "point.lat": 49.7513,
      "point.lon": 18.63213,
      "boundary.circle.lat": 49.7513,
      "boundary.circle.lon": 18.63213,
      "lang": {
        "name": "Polish",
        "iso6391": "pl",
        "iso6393": "pol",
        "defaulted": false
      },
      "querySize": 20
    },
    "engine": {
      "name": "Pelias",
      "author": "Mapzen",
      "version": "1.0"
    },
    "timestamp": 1553680429398
  },
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          18.64089,
          49.747574
        ]
      },
      "properties": {
        "id": "101842315",
        "gid": "whosonfirst:locality:101842315",
        "layer": "locality",
        "source": "whosonfirst",
        "source_id": "101842315",
        "name": "Cieszyn",
```

```
        "confidence": 0.6,
        "distance": 0.755,
        "accuracy": "centroid",
        "country": "Polska",
        "country_gid": "whosonfirst:country:85633723",
        "country_a": "POL",
        "region": "Województwo śląskie",
        "region_gid": "whosonfirst:region:85687277",
        "region_a": "SL",
        "county": "CieszyÅski",
        "county_gid": "whosonfirst:county:102079421",
        "locality": "Cieszyn",
        "locality_gid": "whosonfirst:locality:101842315",
        "continent": "Europa",
        "continent_gid": "whosonfirst:continent:102191581",
        "label": "Cieszyn, Polska"
      },
      "bbox": [
        18.6203222314,
        49.7168028692,
        18.6790328002,
        49.7830528211
      ]
    }
  ],
  "bbox": [
    18.6203222314,
    49.7168028692,
    18.6790328002,
    49.7830528211
  ]
}
```


3. ***Structured*** (beta) - pozwala na wyszukiwanie po konkretnych danych np. Country, czy postalcode.

   Odnajdywanie kraju:

   https://geo.flotea.pl/v1/search/structured?country=POL
   https://geo.flotea.pl/v1/search/structured?country=PL

   Dokumentacja:
   https://github.com/pelias/documentation

Usuwanie danych

- Usunięcie danych z whosonfirst z /sqlite i z /data
- Usunięcie danych z geonames (ściągnięte pliki .zip) - zwykle ściąga się AllCountries.zip w przypadku kiedy następuje próba uruchomienia “całego świata”.
- Uruchomienie skryptu czyszczącego bazę ze /schema i utworzenie nowego schematu (ElasticSearch)

```
# !! WARNING: this will remove all your data from pelias!!
node scripts/drop_index.js
./bin/create_index
```

Uwaga: W przypadku wdrożenia własnego, należy pamiętać, że wyszukiwarka pelias działa w taki sposób, że należy do niej wykonać dwa zapytania /search i /autocomplete. Dzięki czemu uniknie się błędów związanych z wpisywaniem perfekcyjnie nazw miejscowości. W przypadku wpisania np. Wiedeń, autocomplete zwraca podpowiedzi podobne, z wyłączeniem precyzyjnego wyniki. Search za to działa dobrze z precyzyjnymi wynikami.

Alternatywnym rozwiązaniem dla Pelias jest wdrożenie podpowiedzi wyników na podstawie rozwiązań Google Autocomplete, API openstreetmaps. Jednakże rozwiązania te mają ograniczenia: google jest rozwiązaniem płatnym, OpenStreetmaps natomiast jest ograniczone ilością zapytań).

Opcjonalnie: Konfiguracje dla Środowiska testowego PoA Ethereum

Uwaga: Opcjonalne w przypadku chęci tworzenia lub rozwoju kodu kontraktów w prywatnej sieci Blockchain.

Na potrzeby projektu możliwe jest zbudowanie środowiska do obsługi urządzeń o wysokiej wydajności. Instrukcje te zostały udostępnione pod adresem:

https://github.com/flotea-io/eth-node. Jest to środowisko Ethereum Proof-of-Authority z pregenerowowanymi portfelami. Zautomatyzowana ścieżka konfiguracji prywatnego środowiska testowego Ethereum do tworzenia oprogramowania w postaci inteligentnych kontraktów w zamkniętej prywatnej sieci testowej. Zawiera konfigurację sieci Ethereum w postaci Bootnode, Statystyk, Portfela, Potwierdzających przelewy, kilku regenerowanych portfeli. W przypadku wdrożenia projektu zaleca się korzystanie z testowej publicznej sieci Blockchain.

V. Dodatkowe Analizy

Analiza - Technologia FI-WARE POI DataProvider

Projekt	FIWARE POI DP
Krótki opis	Baza danych punktów zainteresowania, standard danych + API wyszukiwania. Projekt FIWARE FI-PPP. Stworzony 2013-2014 na Uniwersytecie w Oulu.
wersja	1.1
opracował	Bartłomiej Blicharski
szerszy opis	POI (Points of interest) Serwer dla technologii aplikacji webowych, który wspiera utrzymywanie informacji pasujących do lokacji / lokalizacji / punktów geograficznych. Udostępnia zapytania bazujących o lokalizację i inne kryteria. Może być konfigurowalny wg. potrzeb i dostosowany do określonego projektu. POI umożliwia łatwe przypisywanie dowolnej informacji do miejsc np. atrakcji turystycznych, serwisów, zdjęć, wideo, kontentu 3D, specjalnych danych dla biznesu, obrazować rzeczy dla gier typu "outdoor". Umożliwia wytwarzanie aplikacji utylizujących wszystkie wymienione możliwości. POI DataProvider (Dostarczanie danych) występuje jako główne źródło danych dla rozszerzonej rzeczywistości (AR) takich jak podstawowe informacje o wybranym POI / lub obiekcie 3D reprezentującym dany POI. Dla aplikacji Internetu rzeczy (IoT) POI odnoszą się do konkretnego urządzenia, które identyfikowane są za pomocą UUID oraz real-virtual POI extension server, który kooperuje w dostarczaniu danych z aplikacji do urządzenia. POI (Points of Interest) Generic Enabler to zestaw serwera webowego, który obsługuje: zapisywanie informacji odnoszących się do miejsc, zapytania według lokalizacji i innych kryteriów, możliwości konfigurowania danych do swoich potrzeb.
Specyfikacja	http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/POI_Data_Provider_Open_API_Specification

Projekt udostępnia:

a. Specyfikacja POI-DP API (Dataprovider API)
b. Implementacja dowodu koncepcji sieci POI-DP,
c. Przykładowa aplikacja POI-DP
d. Sposoby tworzenia i implementowania aplikacji specyficznych POI-DP

Dane z POI mogą być użyteczne w szerokim zakresie w różnego rodzaju aplikacjach. Większość aplikacji jest połączona z turystyką oraz odnajdywaniem czegoś ciekawego w nowych miejscach. POI może być bardzo użyteczny w tego typu aplikacjach np.

a. Dokładne informacje nt. przystani dla aplikacji jachtów, lotniska itp.
b. Dokładne informacje o podziemnych strukturach i elementach.
c. Informacje o geograficznym rozlokowaniu urządzeń, informowaniu o urządzeniach potrzebujących naprawy itp.
d. Hotele, festiwale, miejsca fast-foodów, szpitale, wszystkie informacje, których potrzebuje turysta.

Przykładowy scenariusz użycia:

Use Case: Skipper z jachtem zbliża się do punktu przybycia. Jego zainteresowaniem jest zdobycie informacji na temat przystani do której dobija, chce znaleźć miejsce na kolację oraz hotel w pobliżu przystani.

Przykład komunikacji w data POI Provider

a. Urządzenie mobilne wysyła zapytanie na serwer z punktami zainteresowania
b. Aplikacja w urządzeniu mobilnym - pokazująca interesujące dla skippera punkty zainteresowania na ekranie w aktualnym miejscu. Baza znajdująca się na serwerze nr. 1.

c. Aplikacja turystyczna w urządzeniu mobilnym - aplikacja pozwala na zdobycie danych punktów zainteresowania na mapie. Baza znajdująca się na serwerze nr. 2.
d. Serwer z oprogramowaniem żeglarskim - serwer wykonuje zapytania do serwera POI uzyskując podstawowe informacje o danym POI. Filtruje je dla zainteresowania zgodnego z żeglarskim POI, pobiera informacje nt. POI z detalami, informacjami takimi jak: głębokość przy przystani czy specjalne usługi świadczone na tej przystani.
e. Turystyczny serwer - serwer wykonujący zapytania do serwera POI uzyskując podstawowe informacje przypisane dla danego POI z danymi odnoszącymi się do turystyki tj. np. klasyfikacja (gwiazdki) hoteli, klasyfikacja restauracji itp.
f. Serwer POI - serwer pozwalający utrzymywać bardzo wielką ilość danych POI, wyspecjalizowany w wyszukiwaniu dotyczącym regionu geograficznego.

Różne komponenty POI mogą być zachowywane w różnych bazach danych na różnych serwerach, udostępnianych przez różne organizacje. Te dane są połączone do punktu zainteresowania (POI) za pomocą UUID identyfikującego dany punkt POI. Następnie dane te mogą zostać zebrane ze wszystkich serwerów w jedno miejsce i wyświetlone w odpowiedni sposób (jest tutaj zapewniona unifikacja rozwiązania pod warunkiem, że każdy z tych serwerów dobrze zaimplementował technologię - zgodnie ze standardami)

Przepływ informacji:

a. Skipper używa żeglarskiej aplikacji mobilnej, która pokazuje punkty zainteresowania zgodne z tematyką żeglarską.
b. Żeglarska aplikacja wysyła zapytanie do żeglarskiego serwera POI, który odpytuje główny serwer z punktami zainteresowania. Serwer lub aplikacja mogą zestawić dane z innych serwerów POI np. Informacje o hotelach w pobliżu itp.
c. Żeglarski POI serwer używa identyfikatorów UUID danego punktu zainteresowania do wyszukania informacji, a następnie przesyła je do aplikacji mobilnej.

d. Skipper wybiera interesujący go POI korespondujący z portem do którego przybija, czyta ekstra informacje zebrane z innych serwerów. Aplikacja turystyczna otrzymuje UUID danego punktu POI - punktu zainteresowania w wybranym regionie - np. port w Gdyni. Aplikacja turystyczna / serwer wysyła zapytanie do serwera turystycznego który odpytuje serwer POI na temat pobliskich punktów dotyczących pobliskich hoteli.
e. Serwer turystyczny pobiera turystyczne informacje ze swojej bazy danych oraz przekazuje je do aplikacji mobilnej
f. Skipper może teraz przeglądać i wybierać informacje o interesujących go usługach powiązanych z POI

Rozszerzona Rzeczywistość (Augmented Reality)

POI-DP występuje jako główny zbiornik danych dla rozszerzonej rzeczywistości (AR) - w FIWARE połączony z inną technologią tj. Augmented Reality. Informacje o POI mogą być wizualizowane w widoku AR. Na przykład podstawowe informacje o danym POI lub model 3D reprezentujący dany POI. Można w ten sposób w przyszłości prezentować dane 3D na mapach.

Rzeczywiste wirtualne interakcje (Real Virtual Interaction)

Dla tych aplikacji, które dotyczą “Internetu Rzeczy” (ang. Internet of Things), rzeczywiste/wirtualne informacje mogą zostać dostarczone. Punkt POI identyfikowany przez UUID jest połączony z realnym urządzeniem. Serwery kooperują aby udostępnić informacje o urządzeniu. Ta cecha może pomóc kontrolować urządzenia wykonawcze np. ukierunkować kamerę lub ustawić termostat, czy przeczytać wartości sensorów.

Podstawowe koncepcje

1. Zunifikowany model POI

    a. Proste do rozszerzania. Prosta dystrybucja jako niezależne komponenty danych.
    b. POI jako dodatkowe dane do innych zasobów POI np. restauracje w okrętach kursowych. API pomiędzy wyszukiwarkami mogą być używane wzajemnie.
    c. Możliwa asocjacja pomiędzy danymi 3D a konkretnymi punktami POI. Możliwe asocjacje innych technologii.
    d. Hierarchia punktów POI

2. Webowy serwis udostępniający dane za pomocą API (RESTful). Web serwis umożliwiający odpytywanie, wyciąganie i modyfikację danych punktu POI.
    a. Wyszukiwania limitowane przestrzennie (określony zasięg przestrzeni geograficznej)
    b. Wyszukiwania ograniczone do określonych kategorii np. kolacje, kursy golfa, dentyści, kasyna, muzea itp.
    c. Wyszukiwania i ograniczenia czasowe np. na odnalezienie restauracji, które są obecnie otwarte. Obecnie dostępnych punktów.
    d. POI CRUD (Usuwanie, Czytanie, Aktualizowanie i Usuwanie danych)

Architektura sieci danych punktów zainteresowania

Architektura RESTful jest stworzona po to aby utrzymywać i dostarczać dane POI (Punktów zainteresowania). Serwis udostępnia zróżnicowaną funkcjonalność dla tworzenia zapytań, dostępu i modyfikowania danych POI. Serwisy są przywoływane poprzez RESTful API używając standardu protokołu HTTP. Dane POI mogą być z łatwością przekazywane dzięki modularnej strukturze. Naturalną

konsekwencją jest to, że serwis i jego architektura mogą świetnie dystrybuować dane. Różne serwisy “backend” oparte o POI mogą utrzymywać różne zbiory komponentów POI. Architektura umożliwia również implementowanie kompozycji serwisów np. serwis backendowy, który niekoniecznie utrzymuje własne zestawy POI, ale udostępnia możliwość pobierania punktów z różnych innych serwerów, gdzie są one utrzymywane. Dane POI mogą być dostępne i wyszukiwane poprzez RESTFul API udostępniane przez webserwisy konkretnych aplikacji. Punkt dostępowy do POI umożliwia dostęp do specyficznych komponentów POI bazujących o identyfikator UUID. POI search API udostępnia różne wyszukiwania bazujące na lokalizacji dla klienta, który może wyszukać interesujący Punkt w konkretnej lokalizacji.

Tworzenie zapytań o punkty zainteresowań

1. Wyszukiwanie radialne (wg. odległości od danego punktu)

Główną funkcjonalnością jest wyszukiwanie POI (Punktów zainteresowania) blisko konkretnego wyznaczonego punktu geograficznego np. miejsca w którym znajduje się użytkownik. Może być użyte do odnajdywania miejsc (POI) wg. zadanego dystansu lub promienia okręgu. Lokacje prezentowane są za pomocą punktów geograficznych (latitude, longitude) wyrażonych w stopniach oraz promienia wyznaczonego w metrach.

2. Wyszukiwanie Bounding Box

Wyszukiwanie bounding box umożliwia wyszukiwanie punktów zainteresowań znajdujących się w geograficznym regionie wyrażonego za pomocą prostokąta. Do metody bounding box-a wysyła się parametry odpowiedzialne za określenie tego prostokąta (zwykle wystarczą dwa punkty geograficzne).

3. Dostęp do informacji o POI poprzez unikalny identyfikator UUID.

W innym wypadku klient zna unikalny identyfikator UUID, którym jest zainteresowany. W tym wypadku POI może być dostępny bezpośrednio z POI UUID. Kilka punktów UUID może być przetwarzanych za pomocą jednego zapytania, dzięki czemu możliwe jest wyciągnięcie szczegółowych informacji o tych punktach (np. w przypadku portalu flotea.pl będą to punkty dotyczące przystanków, czy połączeń)

Modyfikowanie punktów zainteresowań

Dodawanie POI - Nowe punkty mogą być dodawane do bazy danych, ich zawartość może być przesłana za pomocą klienta używającego zapytania POST HTTP. Aktualizowanie POI - Istniejące punkty POI mogą być aktualizowane. Aktualizowane dane POI mogą być wysłane za pomocą POST HTTP. Usuwanie POI - Mogą być usunięte z bazy danych. Klient może usunąć istniejący punkt POI używając DELETE HTTP.

Podstawowe zasady

1. Rdzeń punktu zainteresowania musi być kompaktowy, nie może zawierać wielu informacji poza tymi które są potrzebne. To pozwala na utrzymywaniu miejsca oraz na szybkości zapytań i odpowiedzi.
2. Musi być proste - technicznie i organizacyjnie dla aplikacji. Organizacja musi mieć prosty dostęp do rozpoczęcia używania danych z POI i do dodawania potrzebnych rozszerzeń do danych.
3. Organizacja musi posiadać możliwość edycji POI
4. Używanie pomiędzy aplikacjami, klientami i serwerami musi być bez błędów. Oczywiście część danych zostanie utracona jeśli nie użyje się transakcji
5. Puste dane nie są transmitowane
6. Użytkownik może limitować zapytania:
    a. maksymalna ilość POI do zwrócenia
    b. listowanie potrzebnych pól danych
    c. wybór języka dla pól danych

d. dzielenie danych, np. wybór tylko restauracji, które są aktualnie otwarte

Budowa Architektury systemu z zastosowaniem punktów zainteresowania

Architektura systemu bazująca na serwerach POI jest wolna. Istnieje możliwość używania wielu publicznie dostępnych serwerów różnych usług razem. Możliwość wymieniania się danymi. Używanie dystrybuowanych danych, aplikacja zorientowana na POI może posiadać konieczność rozszerzania możliwości dostępnego publicznie serwera POI np.:

a. Więcej danych musi być połączonych z danym POI
b. Więcej POI jest potrzebnych
c. Prywatne rozszerzenia danych danego POI są wymagane
d. Wysoka integracja danych jest potrzebna

Te wymagania mogą być spełnione używając różnych baz danych razem. Prywatne serwery POI z możliwością kontroli dostępu mogą być użyte aby spełniać wymagania krytycznych rozwiązań. Proste kombinacje danych na serwerach POI dla użytku:

a. Program kliencki może odpytywać różne serwery zawierające dane POI
b. Serwer POI może odpytywać inne serwisy POI.

Zapytanie może zawierać:

a. Więcej punktów zainteresowania (POI)
b. Więcej danych danego punktu POI.

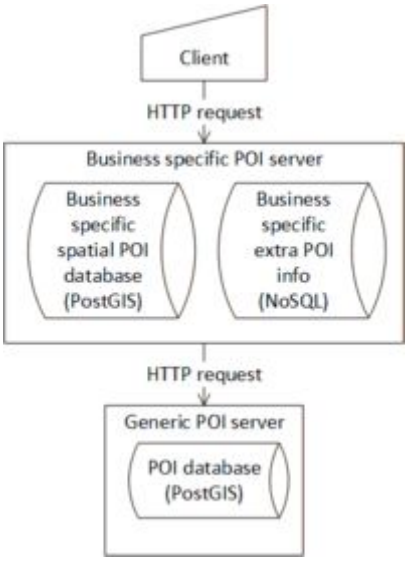
Ponieważ POI są identyfikowane przez unikalne UUID jest możliwe łączenie danych na temat danego punktu zainteresowania z różnych serwerów, również nie powiązanych ze sobą. UUID danego POI powinny być takie same w różnych bazach danych w innym wypadku może się okazać, że będzie wymagane dodatkowe mapowanie danych.

Używanie odseparowanych danych

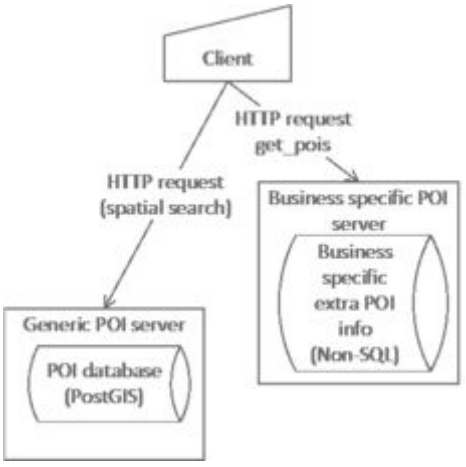
Oczywiście używanie publicznych serwerów POI nie jest wymagane, jeżeli nie są potrzebne dla aplikacji. Aplikacja może używać dowolnej bazy POI oraz totalnie odseparować się od publicznego udostępniania danych.

Baza danych SQL i noSQL

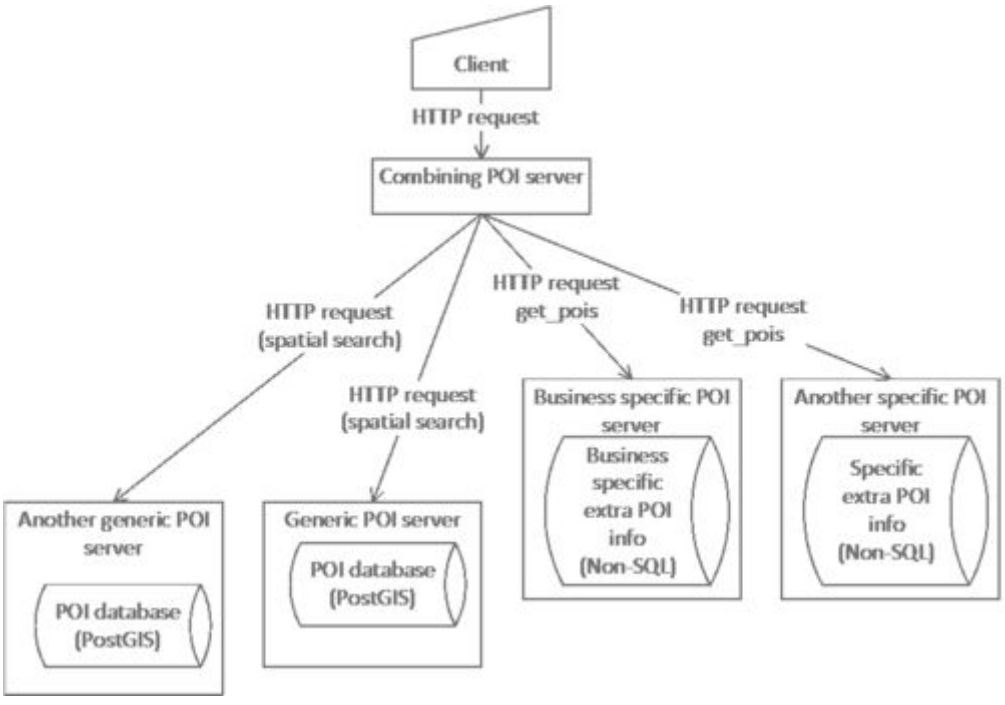
Bazy danych SQL zawierające PostGIS są dobre w wyszukiwaniu danych w oparciu o dane geograficzne, zawierają również specjalny typ danych (geometrie). Bazy danych noSQL są łatwe w użyciu, ale słabe w wyszukiwaniu (w tym danych na bazach geograficznych). Architektura POI używa bazy danych z dodatkiem PostGIS dla operacji opartych o wyszukiwanie zaawansowane. Baza noSQL jest używana do trzymania szczegółowych informacji na temat danego POI. PostGIS udostępnia UUID danego POI, który następnie jest użyty jako klucz w wydobywaniu danych i informacji z bazy NoSQL - w ten sposób uzyskuje się resztę danych.



Klient wysyła zapytanie HTTP do specyficznego serwera POI, gdzie następuje wyszukanie punktów zainteresowania, następnie szczegóły mogą być pobrane z bazy noSQL na tym samym serwerze. Trzeba pamiętać, że serwer może wykonać zapytanie do innego Serwera POI o szczegóły tych danych.



Klient kontaktuje się poprzez HTTP z wyszukiwarką bazującą na położeniach geograficznych, następnie drugim zapytaniem pobiera konkretne dane POI - komunikuje się z serwerem po to aby wydobyć informacje szczegółowe (z noSQL).



Klient komunikuje się z serwerem POI prywatnym, który na podstawie zadanego zapytania może pobrać dane z różnych serwerów POI, oraz informacje dodatkowe z różnych serwerów POI noSQL (kombinując w ten sposób zestawienie różnych danych w jedną całość).

Dostarczanie danych POI z wielu backendów

Możliwe jest napisanie komponentu dostarczającego dane opartego o POI Data Provider który komponuje dane z wielu źródeł i innych POI data providerów (innych backendów). Wysoki poziom logiki aplikacji dla takich rozwiązań są zaprezentowane poniżej:

1. Obsługiwanie zapytań opartych o region (radial search)

    a. Wysłanie zapytania do backendu POI, który umożliwia obsłużenie zapytania (core backend)
    b. Parsowanie zapytania, wykonywanie dodatkowych zapytań dla każdego UUID dostępnego w zapytaniu
    c. Powtórzenie zapytań do reszty POI serwerów backendowych i odesłanie dodatkowych danych dla każdego UUID
    d. Konstrukcja ostatecznej odpowiedzi w formacie JSON zawierającego dane na temat komponentów POI o zadanych UUID.

2. Obsługa zapytań get_pois - zapytanie jest prostsze niż wyszukiwanie po regionie, bezpośrednio można uzyskać listę wybranych UUID, które obsługują następujące akcje:
    a. Przeszukanie wszystkich powiązanych backendów POI i wydobycie dodatkowych informacji używając UUID.
    b. Skonstruowanie finalnej odpowiedzi w formacie JSON oraz dodanie do nich potrzebnych danych.

Programowanie aplikacji klienta:

a. detale komunikacji z serwerami POI
b. struktury danych dostarczane przez serwer POI

Operacje po stronie aplikacji klienta:

a. Sortowanie w zakresie zainteresowania POI, w konkretnym obszarze oraz z konkretnymi parametrami
b. Zapytania o POI używając zapytań opartych o położenie geograficzne oraz innych parametrów - możliwa interakcja użytkownika
c. Wybranie punktu zainteresowania - możliwa interakcja użytkownika.
d. Zapytanie o dodatkowe informacje nt. POI
e. Używanie / pokazywanie danych z POI - możliwa interakcja użytkownika

Wnioski

W rozwiązaniu punktami POI jest trasa, dlatego niemożliwe jest zastosowanie technologii, która opiera się jedynie o pojedynczy POI w kontekście całej trasy. Trasy są zbiorami punktów stałych, a w niektórych przypadkach np. Tras budowanych na żądanie, są budowane dynamicznie w momencie zakupu biletów / rezerwacji przejazdu. W Technologii POI DataProvider występuje jednak bardzo przydatny element dotyczący harmonogramowania tras, który jest wzięty jako wzór do budowania Tras. W projekcie Flotea, który budowanej w latach 2013, pominięto temat optymalizacji harmonogramu tras skupiając się na tzw. Masowym podejściu, żeby tras było jak najwięcej, gdzie każdy element trasy był osobą ofertą. Efektem było wizualne przedstawienie zasobów jako bardzo wiele ofert, jednakże przefiltrowanie lub dotarcie przez pasażera do konkretnej trasy wiązało się z trudnościami, mimo, że miał przedstawioną stosunkowo bogatą ofertę, która była nieskuteczna, skomplikowana i obciążającym systemy informatyczne. Opracowanie skutecznego harmonogramowania, które będzie bardzo wydajnym rozwiązaniem jest

kluczowe dla sukcesu projektu, żeby na małych zasobach systemowych możliwe było wykonanie jak największej ilości operacji.

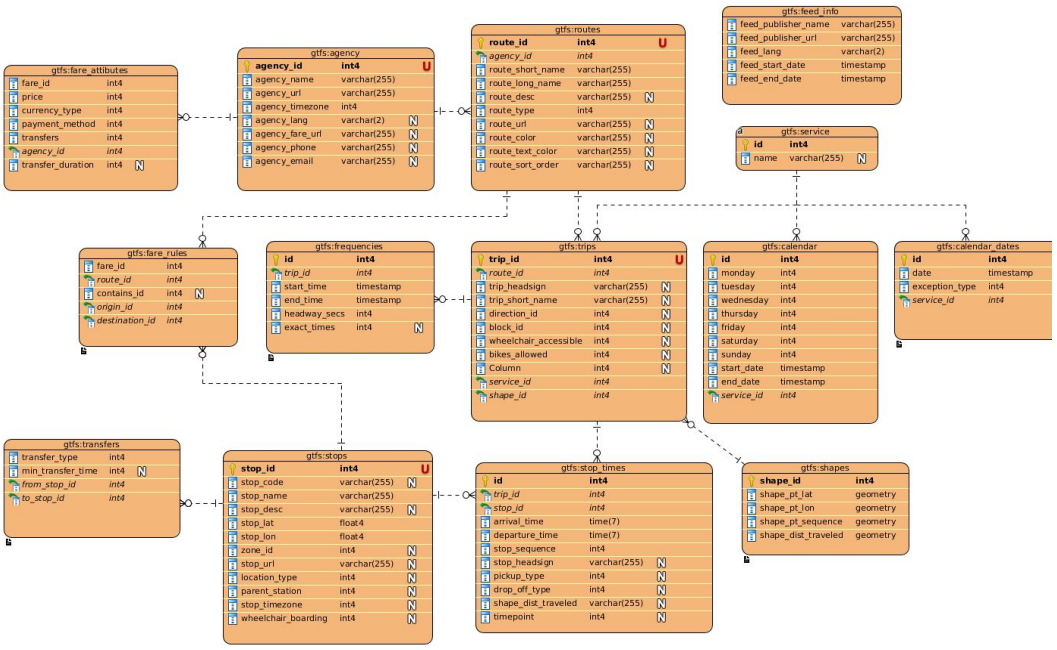
Punkty POI można wykorzystać w przyszłości budując bazę danych przystanków. Zaleceniem do przyszłych analiz jest opracowanie analizy Orion Context Broker, który jest flagowym produktem opensource technologii FiWare na 2020 rok. Projekt Fiware POI DataProvider - nie jest aktualizowany od 2016 roku, podczas dla samych punktów POI statycznych i dynamicznych o dowolnej strukturze Context Broker wydaje się być odpowiednim zamiennikiem.

Analiza - Model Bazy danych w oparciu o FI-WARE / GTFS i NGSI

Dla transportu technologia FI-WARE opisuje zastosowanie modelu GTFS - jest Międzynarodowym formatem zapisu informacji o rozkładach jazdy i lokalizacji przystanków. Znany również jako standard statycznego tranzytu. Definiuje format dla harmonogramowania publicznego transportu. Umożliwia publicznym agencjom transportowym publikować dane tranzytowe i umożliwia programistom pisać aplikacje w oparciu o te dane. FI-WARE opisuje dokument jak mapować standard GTFS do danych NGSI FI-WARE, opisuje również modele stosowane w transporcie do: monitorowania przepływu ruchu, prywatnych pojazdów, publicznych pojazdów (busy, pociągi, itp.), pojazdów komunalnych (np. śmieciarki), specjalnych pojazdów (ambulansy, wozy strażackie, itp.). Zdefiniowane byty transportowe to:

a) TrafficFlowObserver - reprezentujący obserwacje nt. przepływu ruchu.
b) Drogi - zawiera geograficzne i kontekstowe opisy dróg
c) Segment Drogi - zawiera geograficzne i kontekstowe opisy segmentów dróg
d) Pojazdy - reprezentuje pojazd z pełną charakterystyką
e) Model Pojazdu - reprezentuje model pojazdu, statyczne właściwości takie jak: kierunki jazdy, materiały i funkcjonalności.

Poniższy rysunek obrazuje ogólnie jak wygląda struktura modelu danych GTFS (dostępna w plikach źródłowych projektu).



W GTSF istnieje pojęcie taryf/opłat, które można obsługiwać i przypadków ich użycia. Przykłady Taryf / Opłat:

1) Wszystkie podróże mają tą samą taryfę / opłatę, nielimitowane transfery

Np. Agencja przewożąca posiada tak skonsturowaną strukturę opłat:

- 1 USD (price 1 currency USD, payment method 0). Bilet jest dla wszystkich pojazdów i nie wygasa (transfers="" (empty)). Pasażerowie mogą jeździć jak daleko chcą (nieustawione transfer duration). Dopóki wszystkie trasy posiadają tą samą taryfę - Przewoźnik może ominąć wypełnianie fare_rules.
- Planer tras GTFS oblicza taryfę dla dorosłych: w tym przypadku Adult Fare wyniesie $1, gdy pasażer przejdzie do innego pojazdu

2) Wszystkie podróże mają tą samą taryfę, nie posiadają transferów

- $1 USD, payment_method na 0 (pasażerowie mogą jeździć jak długo chcą / jak daleko chcą). Transfer_duration jest pominięty, każda zmiana pojazdu wymaga nowej opłaty (transfers=”0”). Dopóki wszystkie przejazdy posiadają taką samą taryfę, Agencja może pominąc uzupełnianie fare_rules.
- Planer tras GTFS oblicza taryfę dla dorosłych: $1 dolar. W przypadku zmiany pojazdu / przesiadki $2

3) Wszystkie przejazdy posiadają tą samą taryfę, transfery są “dostępne”

- Agencja ustala 1 USD za wejście do pojazdu, payment_method ustawia na 0. Ustawienie transfers na “”, transfer_duration na “5400” - ustala, że nielimiotowane wsiadki przesiadki są dostępne przez 90 minut (5400). Np. bilet na autobus miejski.
- Planer tras GTFS oblicza taryfę dla dorosłych: 1 USD używany przez 90 minut. W przypadku użycia biletu po upływie 90 minut kolejny 1$.

4) Różne ceny dla lokalnych i różne dla ekspresowych tras (podział na lokalne i ekspresowe przejazdy)

Agencja transportowa posiada następujące ustawienia:

- 1.75 USD na lokalne pojazdy (trasa 1), 5 USD na wejście do pojazdów typu ekspress (trasa 2 i 3). Transfery nie są dostępne.
- Dopóki istnieją trasy, które kosztują więcej niż reszta, Przewoźnik musi użyć pliku fare_rules. Każda trasa powinna posiadać wpisy dotyczące danych cen
- Obliczenie GTFS polega na zastosowaniu stawki 5 USD dla trasy 2 i 3, natomiast dla trasy 1 zostanie zastosowana stawka 1.75. Gdy pasażer będzie chciał jechać trasą 1 i 2, zapłaci 6.75 USD.

5) Zakup transferu - podnosi taryfę / opłatę.

Agencja transportowa posiada następujące ustawienia:

- 1.75 USD na lokalne busy, gdy następuje przesiadka pobierana jest dodatkowo 0.25 USD za zakup “transferu”. Transfery są aktywne przez 90 minut.
- GTFS wybiera zawsze najtańszą obowiązującą taryfę. W przypadku planu podróży który nie ma transferów.
- GTFS z jednym transferem stawka kosztuje 3.50 USD. W przypadku zakupu transferu 2.00 USD. Przewoźnik może ogłosić kwotę w wys 2 USD za wszystkie trasy które wymagają zmiany pojazdu. W przypadku kiedy plan podróży nie wymaga przesiadki - opłata wynosi 1,75 USD.

6) Opłaty zależą od “par stacji”

Agencja transportowa posiada następujące ustawienia:

- W tym przykładzie ważny jest punkt rozpoczęcia i zakończenia trasy
- W tym przykładzie każda stacja musi posiadać identyfikator regionu (zoneID) - w tym wypadku każda stacja jest traktowana jako osobny region.
- Plik fare_attributes, fare rules definiują jedno pole dla każdej pary stacji
- W fare_attributes jest ustalony punkt stacji poczatkowej i celowej (identyfikowanych rowniez po zone_id).
- Tutaj Trip planner GTFS wyszukuje cen połączeń w stacjach.

7) Opłaty zależą od Terytorium (regionów)

Agencja transportowa posiada następujące ustawienia:

- Koncentruje trzy różne regiony (zone). Opłaty zależą od regionu który jest obsługiwany.

- fare_attributes i fare_rules posiadają linie wszystkich możliwych kombinacji “regionów”.

Przykład:

fare_attributes:

fare_id	price	currency_type	payment_method	transfers
F1	4.15	USD	0	“”
F2	2.20	USD	0	“”
F3	2.20	USD	0	“”
F4	2.95	USD	0	“”
F5	1.25	USD	0	“”
F6	1.95	USD	0	“”
F7	1.95	USD	0	“”

fare_rules

fare_id	cointains_id
F1	1
F1	2
F1	3
F2	1
F2	2
F3	1
F3	3
F4	2
F4	3
F5	1
F6	2
F7	3

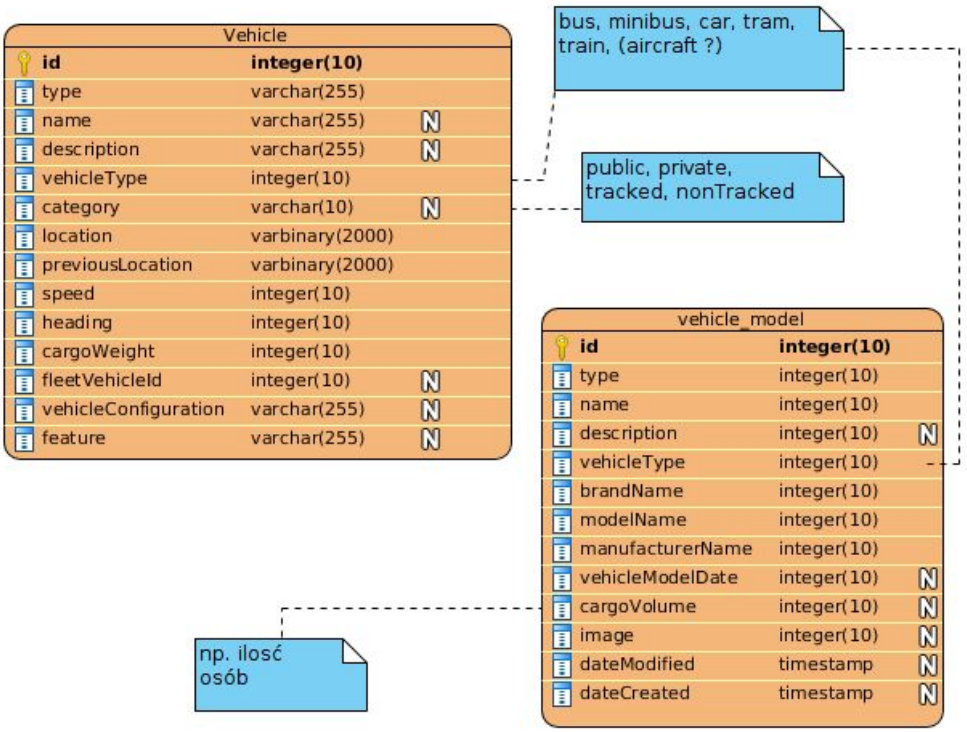
- Wszystkie opłaty przechodzą przez regiony (1,2,3). F2 jest zastosowana do każdej podróży (trip), która przechodzi przez regionu

(1,2). F3 jest zastosowany do każdej podróży która przechodzi przez regiony (1,3) itd.

- Z przejazdu pomiędzy regionami 2 i 3 planer tras wyliczy 2.95 USD.

Przykładowy model Pojazdu z NGSI

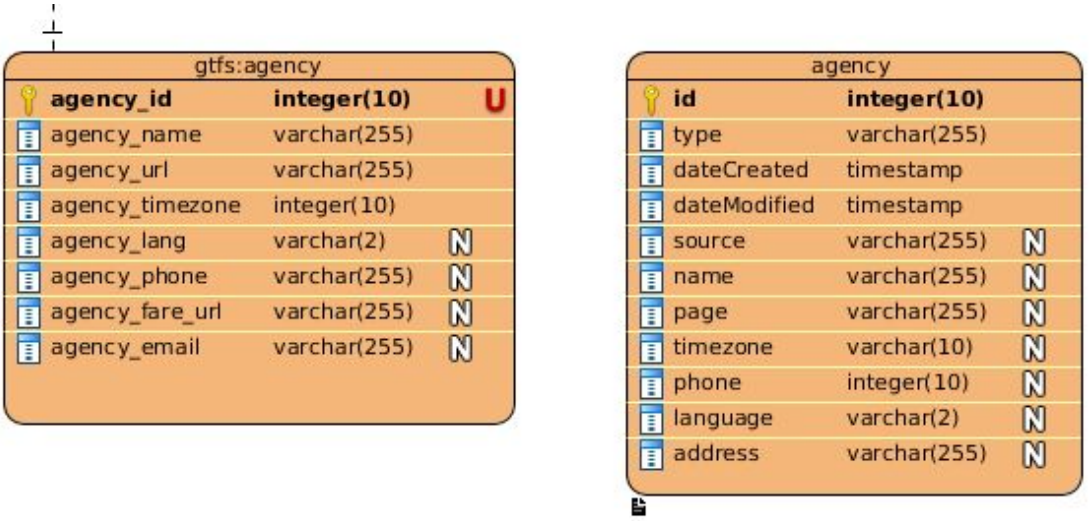
Pojazdy w NGSI dzielą się na pojazdy i model pojazdu. Modele pojazdu są przygotowane w taki sposób, który pozwala na przyjmowanie danych dynamicznie (np. Zmiany położeń geograficznych).



Szczegółowe opisy modeli: GTFS i FIWARE (NGSI)

1. Model Agencji

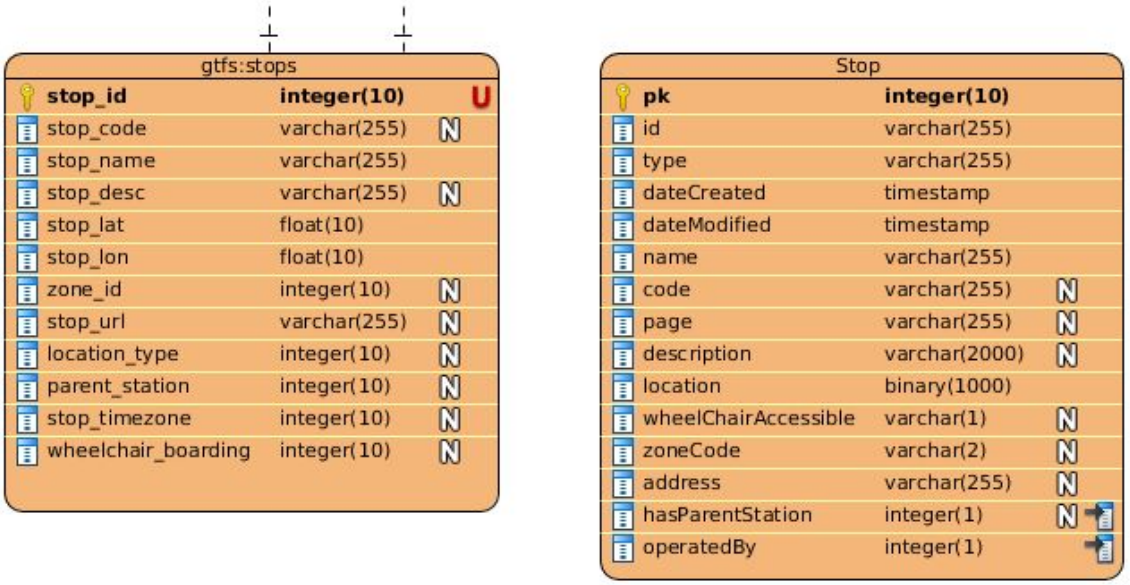
Po lewej stronie model Agencji wg. Standardu GTFS, po prawej wg. Standardu FI-WARE.



Przykład:

```
    {
  "id": 1,
  "type": "gtfs:Agency",
  "name": "Przewoznik Transportowy",
  "page": "http://www.przewoznik.pl/",
  "timezone": "Europe/Warsaw",
  "language": "PL",
  "Source": "http://busy-do-polski.pl/24e86888-b91e-45bf-a48c"
}
```

2. Model Przystanku - jest to typ przystanku, gdzie typ lokacji GTFS:STOP jest równa 0 (należące do danego przewoźnika / dodane przez przewoźnika)



Pola przystanku:

a) pk - klucz - integer(10)
b) typ - stała
c) id - identyfikator - nazwa identyfikacyjna przystanku
d) dateCreated i Modified - pola automatycznie wypełniane
e) Name - nazwa przystanku
f) Code - kod przystanku
g) Page - adres URL przystanku np. https://przystanki.com/uuid
h) Description - Opis przystanku
i) Location - GEO JSON, z punktem umiejscowienia przystanku. (lon,lat)
j) wheelChairAccessible - oznaczenie w jaki sposób jest dostępny dla niepełnosprawnych
k) zoneCode - kod timezone
l) Address - nazwa adres przystanku
m) Klucz obcy - czy ma “nadrzędny przystanek”
n) operatedBy - odnośnik do agencji / nazwa agencji ?

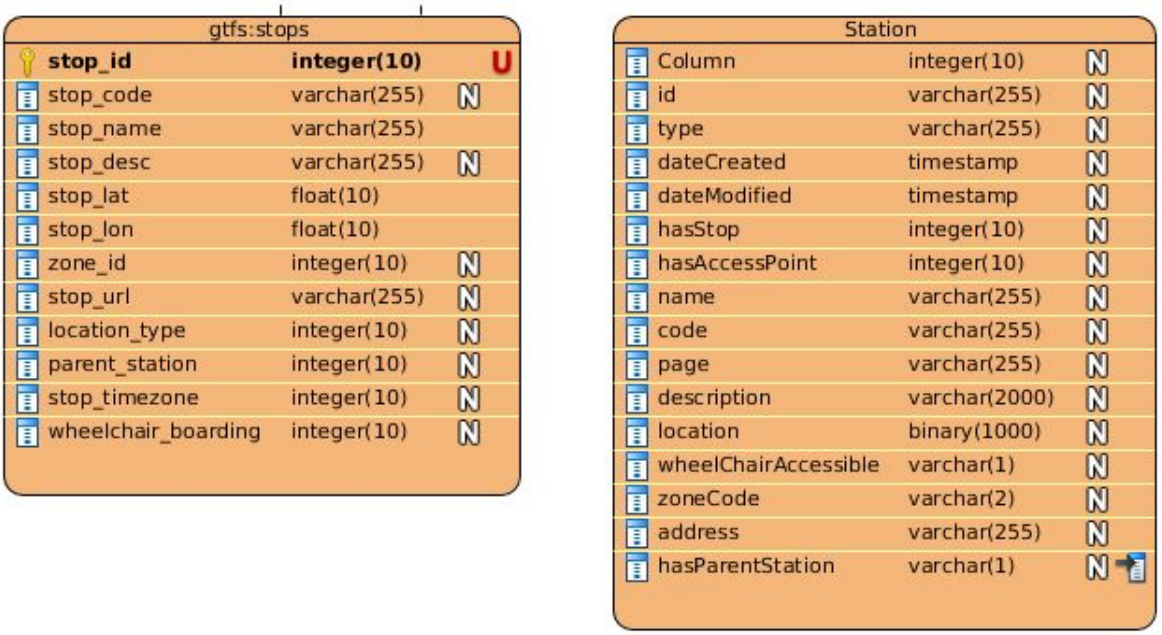
Relacje:

a) parent_station, hasParentStation - powinny wskazywać na inną stację
b) operatedBy - powinny wskazywać na agencję, która dodała/zarządza przystankiem

Przykład:

```
{
  "id": 1,
  "type": "gtfs:Stop", //stała
  "code": "101",
  "name": “Przystanek numer 101",
  "location": {
    "type": "Point",
    "coordinates": [4.424393,36.716872]
  },
  "operatedBy": 1
}
```

3. Model Station - Model stacji gdzie GTFS typ lokacji jest równy 1. W modelach FIWARE jest to rozbicie na 2 modele (Station i Stop). Część pól jest taka sama jak dla Przystanku.



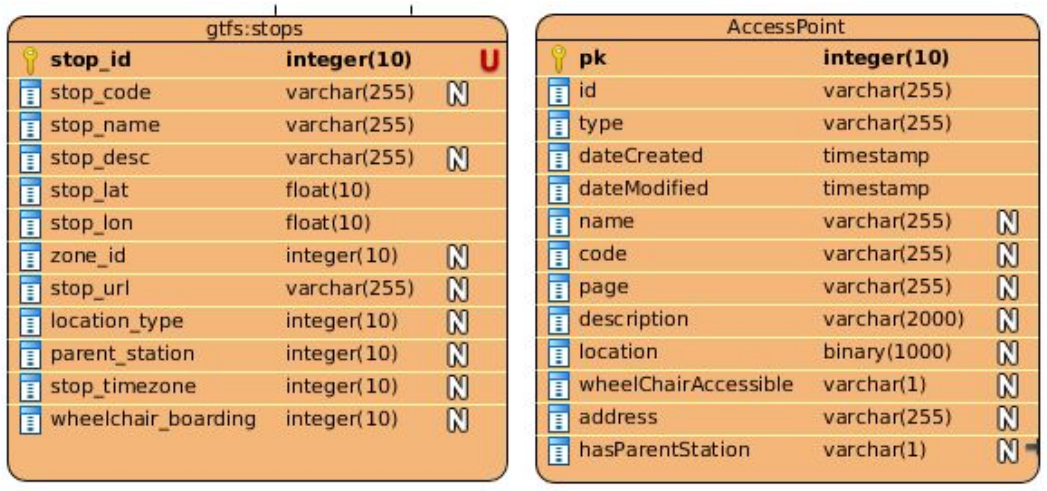
Relacje:

a) hasStop - wskazuje na inny model przystanku (Stop)

b) hasAccessPoint - wskazuje na model typu AccessPoint

c) Wskazuje na model Stacji nadrzędnej.

4. Model AccessPoint - występuje gdy model GTFS:STOP location_type ma wartość równą 2 (jest to wejście na stację, wjazd, itp)



Relacje:

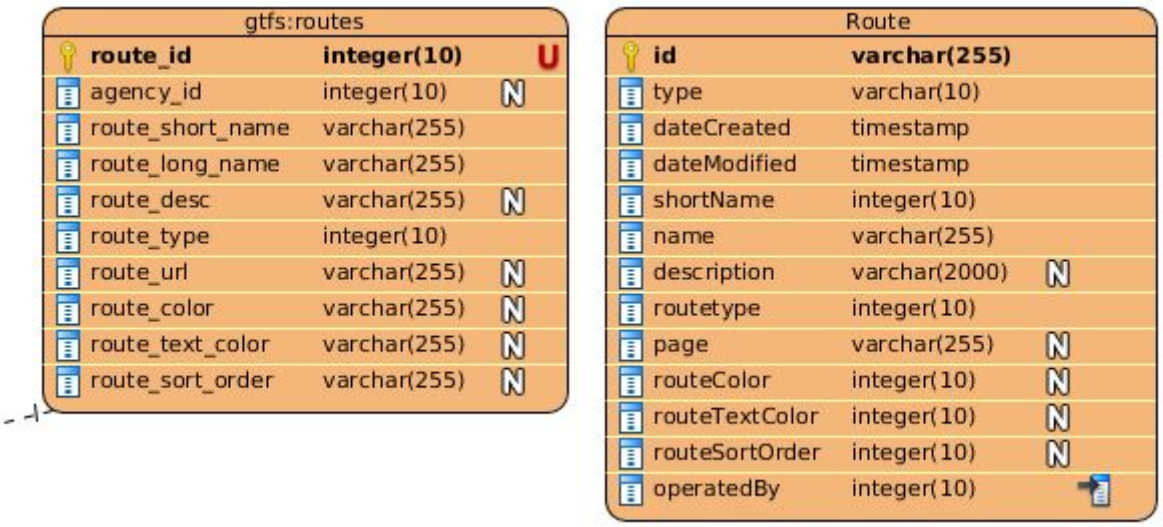
a) hasParentStation - odnośnik do stacji nadrzędnej

Uwagi: Przystanek, Stacja oraz Punkt dostępowy do stacji można rozumieć jako strukturę złożoną gdzie Stacja posiada Przystanki/Przystanek, gdzie Stacja posiada punkt dostępowy do stacji (lub nie). Stacja zatem jest elementem nadrzędnym dla Punktu dostępowego i Przystanków (na stacji może być wiele punktów dostępowych jak i przystanków). Przystanek może należeć do stacji.

Przykład:

```
{
  "id": 1,
  "type": "gtfs:AccessPoint",
  "name": "Wejscie do garazu",
  "location": {
    "type": "Point",
    "coordinates": [-3.69036,40.46629]
  },
  "address": {
    "type": "PostalAddress",
    "streetAddress": "Zamkowa 3 abc",
    "addressLocality": "Cieszyn",
    "addressCountry": "PL"
  },
  "hasParentStation": NULL
}
```

5. Model Trasy (Route)



Pola tras:

a) Id - identyfikator, klucz
b) dateCreated i Modified - automatycznie uzupełniane
c) shortName - krótka nazwa
d) Name - nazwa
e) Description - szczegółowy opis trasy
f) routeType - typ trasy:

0 - Tramwaj, samochód

1 - Metro, kolej podziemna - i inne podziemne typy transportu

2 - Pociąg - pojazdy długodystansowe

3 - Bus - krótko i długodystansowe busy

4 - Łodzie / Statki

5 - Pojazd typu np. Trolejbus (zasilany kablem)

6 - Gondola itp.

7 - Kolej linowa

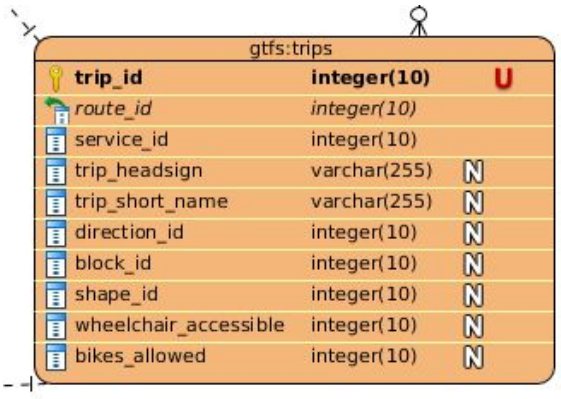
Relacje:

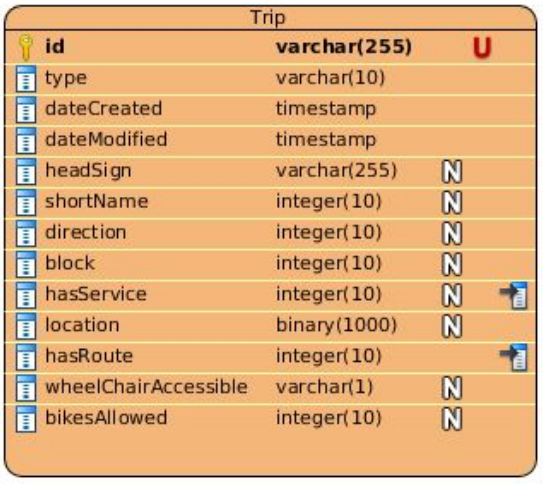
a) operatedBy - relacja do agencji, która obsługuje / wytwarza trasę.

Przykład:

```
{
  "id": 1,
  "type": "gtfs:Route",
  "shortName": "1",
  "name": "Pierwsza traska z Cieszyna do Bielska",
  "page": "http://flotea.pl/cieszyn-bielsko.html",
  "routeType": "3",
  "operatedBy": 1
}
```

6. Przejazd (Trip)

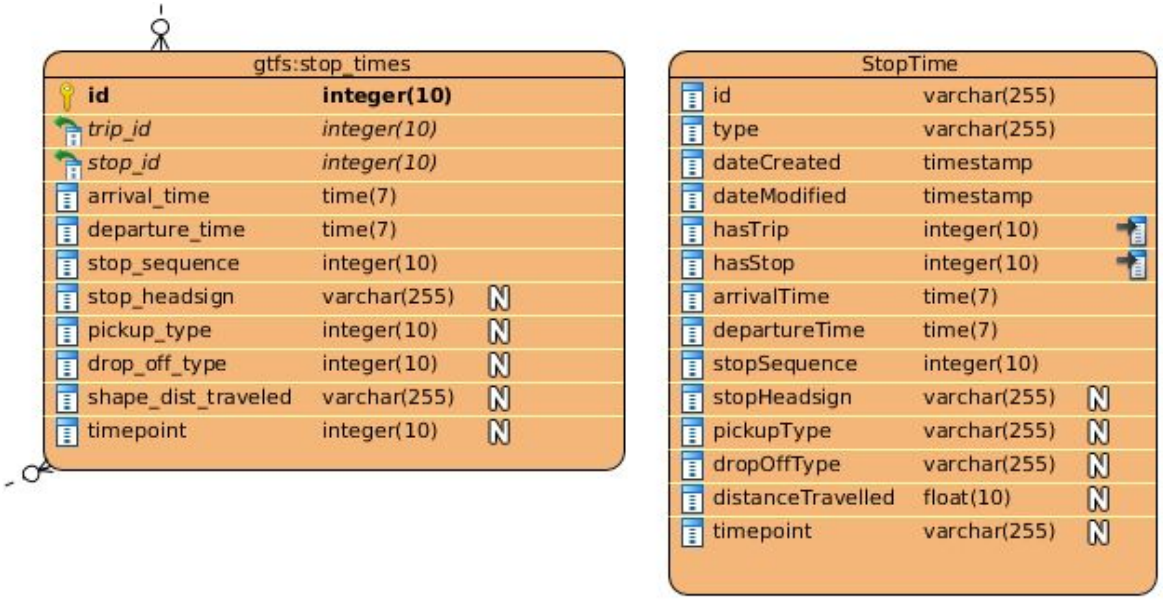





```
{
  "id": "1",
  "type": "gtfs:Trip",
  "hasService": "1",
  "headSign": "Przejazd do Bielska",
  "direction": "0",
  "hasRoute": "1",
  "location": {
    "type": "LineString",
    "coordinates": [
                [-4.421394,36.73826],
                [-4.421428,36.73825],
```

```
                [-4.421505,36.738186],
                [-4.421525,36.738033]
    ]
  }
}
```

7. Model StopTime



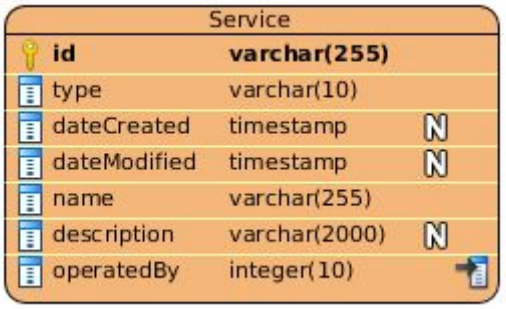

```
{
  "id": "1",
  "type": "gtfs:StopTime",
  "hasStop": "1",
  "hasTrip": "1",
  "distanceTravelled": 759,
  "stopSequence": 4,
  "arrivalTime": "07:04:24",
  "departureTime": "07:04:24"
}
```

Relacje:

a) hasStop - relacja do przystanku

b) hasTrip - relacja do wyjazdu

8. Model Service - reprezentuje usługę transportową, która jest dostępna dla jednej lub więcej tras w określonych datach.

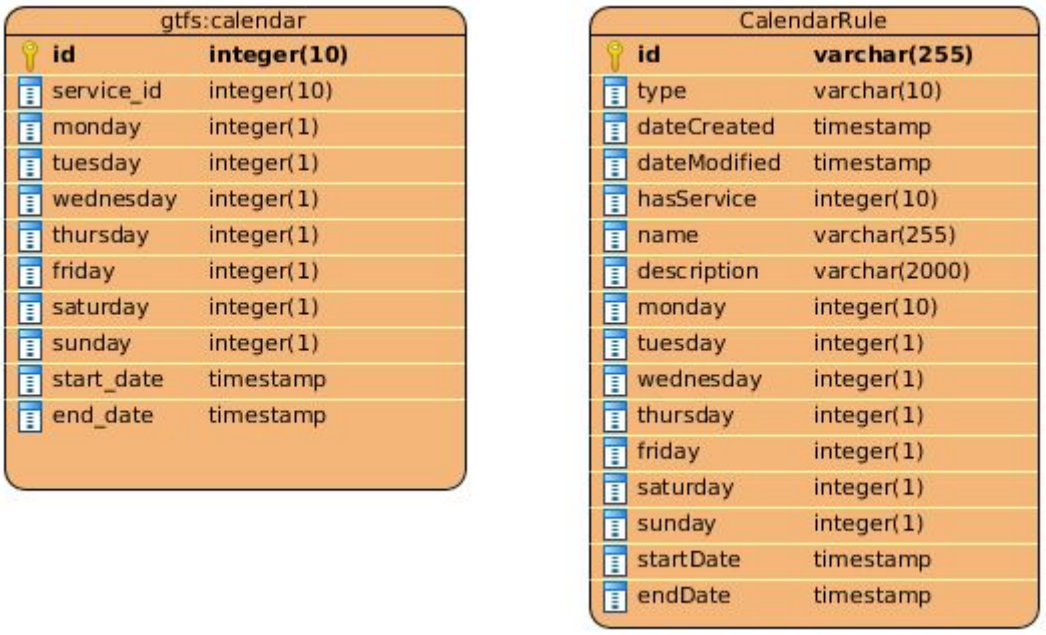



```
{
  "id": "1",
  "type": "gtfs:Service",
  "name": "LAB",
  "description": "Laborables",
  "operatedBy": "1"
}
```

Relacje:

a) OperatedBy - odniesienie do przewoźnika

9. Model CalendarRule



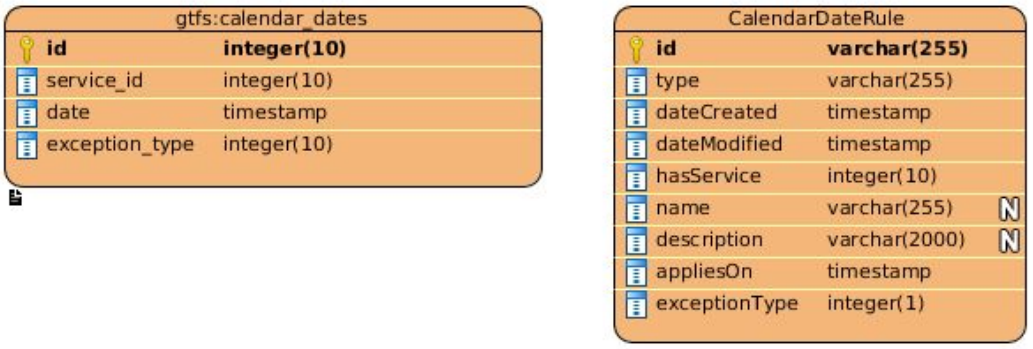
Relacje:

a) hasService - relacja do usługi trasy

Przykład:

```
{
  "id": "1",
  "type": "gtfs:CalendarRule",
  "name": "Service 1 - od Poniedziałku do Piątku przez cały rok",
  "hasService": "1",
  "monday": true,
  "tuesday": true,
  "wednesday": true,
  "thursday": true,
  "friday": true,
  "saturday": false,
  "sunday": false,
  "startDate": "2018-01-01",
  "endDate": "2019-01-01"
```

10. Model CalendarDateRule - aktywowanie lub dezaktywowanie serwisów poprzez datę (np. święta). Można używać w dwojaki sposób, albo doprecyzować daty w modelu kalendarza, albo dodać wyjątki w tym modelu (CalendarDateRule).



ExceptionType - włącza / wyłącza serwis w danym zakresie dat

a) 1 - Usługa transportu powinna działać w wyznaczonej dacie (wyjątkowa data poza kalendarzem dat)
b) 2 - Usługa transportu powinna być wyłączona w wyznaczonej dacie.

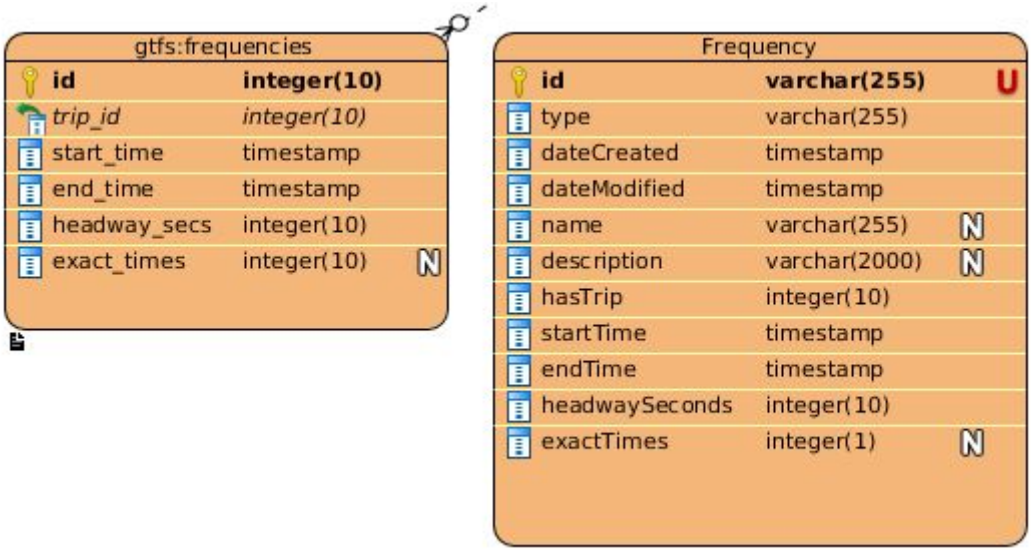
Przykład:

```
{
  "id": "1",
  "type": "gtfs:CalendarDateRule",
  "name": "Rule Fair Area",
  "hasService": "1",
  "appliesOn": "2018-03-19",
  "exceptionType": "1"
}
```

Relacje:

a) hasService - relacja do serwisu

11. Model Frequency - reprezentuje rozkłady jazdy, które nie posiadają ustalonych na przystankach “czasów wyjazdu” i “przyjazdu”. Kiedy przejazdy są zdefiniowane w tym modelu, planer wyjazdów ignoruje wartości “arrival_time” i “departure time” w przystankach (modelu Stops). Kiedy nie ma określonych wpisów we frekwencjach wyjazdów.



a) Headway_secs - pole określające czas pomiędzy wyjazdami z tego samego przystanku, np. A) 05:00:00, 07:00:00, 600, B) 07:00:00, 12:00:00, 1200

b) exactTimes określa czy połączenina bazujące na frekwencjach powinny być wykonywane według informacji wyspecyfikowanych:

0 lub empty - nie są dokładnie wyspecyfikowane (default)

1 - są dokładnie wyspecyfikowane. Są określone wzorem:

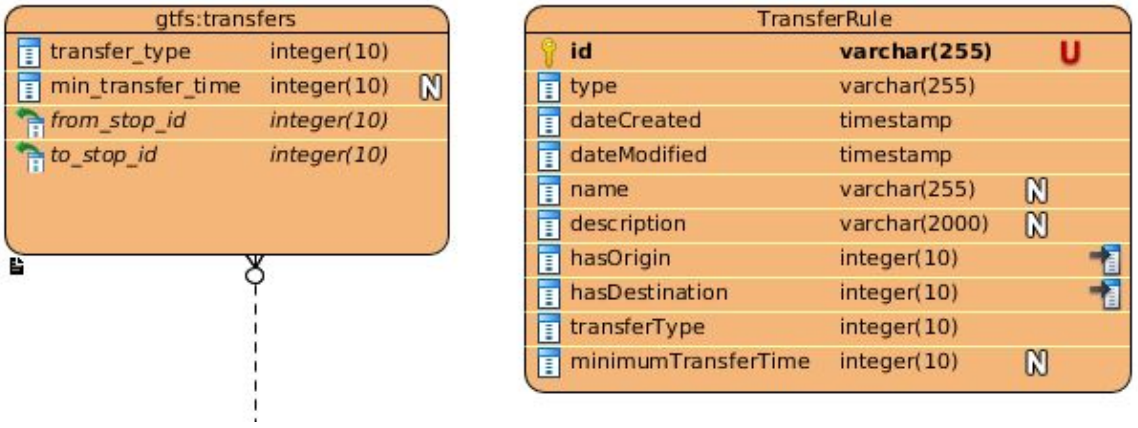
```
Trip_start_time = start_time + x*headway_secs (x = 0,1,2...),
gdzie trip_start_time < end_time
```

Relacje:

a) hasTrip - relacja do podróży

12. Model Transfers - planowanie tras automatycznie kalkuluje transfer pomiędzy trasami w każdej trasie. Transfery należy rozumieć jako opcjonalne

wytworzenie połączeń pomiędzy dwoma trasami (np. Przewóz pasażerów z punktu A trasy A do punktu A trasy B). Przykład: np. Przewóz busem pomiędzy trasami autokarów / przesiadki pasażerów itp.




```
{
  "id": "1",
  "type": "gtfs:TransferRule",
  "name": "L1_L5",
  "hasOrigin": 1,
  "hasDestination": 1,
  "transferType": "0",
  "minimumTransferTime": 10
}
```

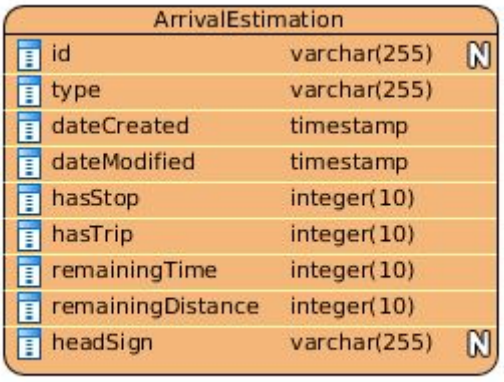
TransferType - wyspecyfikowanie typu połączenia pomiędzy parą przystanków.

- 0 lub empty - rekomendowany transfer pomiędzy punktami dwóch różnych tras
- 1 - Czasowy transfer pomiędzy dwoma trasami. Pojazd powinien czekać na kolejny pojazd (np. Wymiana pasażerów).
- 2 - transfer wymaga minimalnego czasu pomiędzy przyjazdami i odjazdami tras. Określenie minimalnego czasu transferu jest wymagane dla tego typu punktu.
- 3 - transfery nie są możliwe pomiędzy trasami w tej lokalizacji

Relacje:

a) hasOrigin - połączenie do przystanku lub stacji
b) hasDestination - połączenie do przystanku lub stacji

13. Model ArrivalEstimation (nie ma tego w GTFS) - odbiera informacje nt. Czasu przyjazdu na konkretny przystanek pojazdu.




```
{
  "id": 1,
  "type": "ArrivalEstimation",
  "hasStop": 1,
  "hasTrip": 1,
  "remainingTime": "PT8M5S",
  "remainingDistance": 1200,
  "headSign": "Kierunek Cieszyn"
}
```

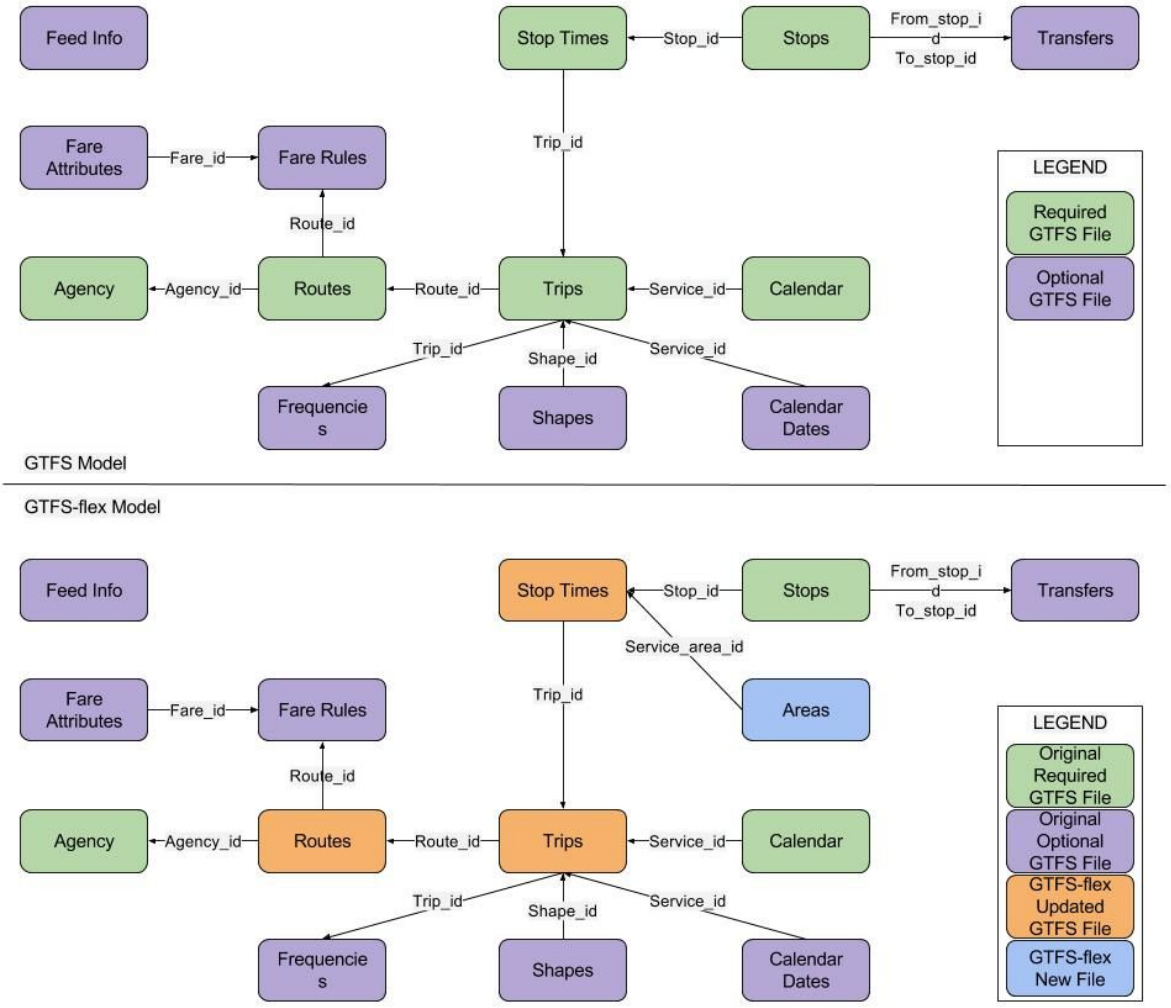
Relacje:

8. hasStop - relacja do przystanku
9. hasTrip - relacja do przejazdu

Wnioski: GTFS stanowi solidną podstawę do budowy struktury bazy danych, baza danych powinna być w jak największym stopniu oparta o ten model danych. Dopuszcza się drobne modyfikacje w celu zapewnienia innych funkcjonalności takich jak zastosowanie harmonogramu wyjazdów czy zapewnienie. Również można dodać klucze obce, aby usprawnić działanie tej bazy w użytej technologii bazodanowej.

Analiza - Rozszerzenie Modelu Bazy danych GTFS Flexible

GTFS Flexible jest prototypem - dodatkiem do GTFS dodającym funkcjonalność transportu na żądanie. GTFS obsługuje jedynie trasy z góry ustalone, model DRT demand-responsive Transportation. Poniższy schemat to porównanie modelu GTFS i modelu GTFS Flexible, kolorami zostały oznaczone zmiany. Generalnie chodzi o to, że GTFS flexible dodaje możliwość używania regionów z których można zamówić przejazd (stąd tabela Areas - nowy element GTFS Flexible). Konieczne są również zmiany w innych tabelach (Trips, Routes i Stop Times).



Model GTFS Flexible powstał po to, żeby wspomóc lepszą obsługę dostosowujących się dynamicznych tras tworzonych z przystanków generowanych pewnymi momentami na żądanie (np. Podczas gdy pasażer nagle zadeklarował chęć przejazdu z danego punktu A do danego punktu B na żądanie), pasażer w takim przypadku wyznacza przystanek, który istnieje tylko przez określony czas.

- Odchylenia trasy (Route deviation) - pojazdy pracujące na trasach regularnych wzdłuż ściśle zdefiniowanej trasy z przystankami lub bez oznaczonych przystanków. To odchylenie serwuje możliwość stosowania transportu na żądanie z określonego obszaru wzdłuż danej trasy. Szerokość lub rozszerzenie obszaru może być ustalone z góry (np. Regionem geograficznym) lub ustalany dynamicznie.
- Odchylenie punktu (Point Deviation) - pojazdy serwujące transport na żądanie w określonym obszarze również oferują limitowaną ilość przystanków (np. Określoną ilością wolnych miejsc w pojeździe). Przystanki te nie posiadają, żadnej z góry określonej regularnej trasy pomiędzy tymi przystankami.
- Konektor reagujący na żądanie - pojazdy operujące w trybie na żądanie w określonym obszarze, w których istnieją jeden lub wiele punktów transferowych - połączonych z trasą standardową (ustaloną z góry) - stałą trasą. Np. zbieranie pasażerów z określonych obszarów i dowóz na stację kolejową lub lotnisko, gdzie rozpoczyna się trasa np. Innego przewoźnika.
- Przystanki na żądanie (Request Stops) - pojazdy poruszające się po statycznych trasach (ułożonych przystanek po przystanku - w z góry zdefiniowanych miejscach) o stałym harmonogramie, które mogą również udostępniać limitowaną ilość przystanków niezdefiniowanych wzdłuż trasy na życzenie pasażera.
- Elastyczne fragmenty tras - pojazdy obsługujące zwykłe trasy, zwykłe harmonogramy przejazdów pomiędzy przystankami, jednakże z możliwością przełączenia w tryb operowania na trasach “na żądanie” (dla limitowanego odcinka trasy).

- Trasa oparta o region - pojazdy obsługujące obszary z przystankami na żądanie wzdłuż korytarza ze stabilnymi punktami wyjazdu i przyjazdu na jednym lub większej ilości punktów w regionie.

Raport TCRP przedstawia liczbę agencji transportowych w USA oceniający jak agencję używają elastycznych usług w transporcie publicznym. Co ciekawe, agencje używają różnych sposobów i technik, używają różnych strategii, kiedy implementują własne systemy do obsługi tego typu tras. Udostępnienie wszystkich sposobów w GTFS może być trudne, jednakże nie jest niemożliwe.

Przykłady użycia:

1. Serwisy związane z odchylaniem tras - pojazd porusza się po stałej trasie i poukładanym harmonogramie przystankowym, może zboczyć z trasy aby wypuścić pasażera (np. Doorstep Service)
2. Serwisy Punkt-Obszar - Kierowca może podjąć pasażera ze statycznego punktu np. Przystanek, stacja. Kierowca może podjąć również wzdłuż danego regionu na którym rysuje się dana trasa (nieposegregowane stacje). Kierowca obsługuje jedynie trasy na żądanie.
3. Odchylenie punktu lub punkty sprawdzające (checkpoint) - kierowca może podjąć pasażera z przystanku, z obszaru wzdłuż trasy (nieposegregowane przystanki) lub w odwrotnej kolejności. Kierowca obsługuje punkty jedynie kiedy żądanie jest ogłoszone przez pasażera.
4. Hail-and-ride - pojazdy poruszają się wzdłuż konkretnej trasy. Pasażer może zgłosić chęć przejazdu na żądanie jedynie na trasie danego pojazdu.

Modyfikacje GTFS

File Name	**State**	**Defines**
`location_groups.txt`	Dodane	Grupy lokalizacji są grupami przystanków i lokalizacji opisanymi GeoJSON (są używane w stop_times)

`locations.geojson`	Dodane	Lokalizacje prezentowane za pomocą GeoJSON które są typami `LineString`, `MultiLineString`, `Polygon` i `MultiPolygon` które określają obszary z których można podejmować pasażerów (z których może zostać zgłoszone żądanie przejazdu)
`trip.txt`	Extended	
`stop_times.txt`	Extended	

Definicje zmian

location_groups.txt (dodane)

Field Name	**Details**
`location_group_id`	(ID, wymagane) Definiuje identyfikator grupy lokalizacji. Grupa jest grupą przystanków i/lub lokalizacji GeoJSON, które razem określają zbiór lokalizacji gdzie pasażer może zgłosić żądanie przejazdu, lub zgłosić żądanie przystanku końcowego. Przystanek w tabeli `stops`nie może mieć takiego samego `stop_id`jak `location_group_id`. Identyfikator w `locations.geojson`, nie może posiadać takiej samej wartości jak `location_group_id`. Wiele wierszy w `location_group` mogą posiadać takie same `location_group_id`.
`location_id`	(ID, Opcjonalne) Dodaje ten przystanek lub georegion GeoJSON do grupy lokalizacji. Referencje: - `stop_id` ze `stops` z `location_type=0`. - `id` lokalizacji w `locations.geojson`.
`location_group_name`	(Text, opcjonalnie) Definiuje nazwę grupy lokalizacji. Jeśli zdefiniowane w wielu wpisach, wszystkie group_name powinny zawierać tą samą wartość lub tylko jedna wartość np. W pierwszym przystanku w grupie mogłaby mieć nazwę.

Locations.geojson - plik / tabela z regionami

- Plik lub baza używająca zestawu georegionów w formacie GeoJSON (RFC 7946)
- Plik musi zawierać typ `FeatureCollection`
- `FeatureCollection` definiuje różne lokalizacje i lokacje z których przewoźnik może obsłużyć zgłoszenia przejazdów na żądanie.

- Tylko typy `LineString`, `MultiLineString`, `Polygon` i `MultiPolygon` mogą być używane. Indywidualne przystanki powinny być realizowane w `stops.txt` nie w `locations.geojson`.
- Każdy GeoJSON `Feature` musi posiadać `id`. Lokalizacja w `stops.txt` lub lokalizacja w `location_groups.txt` nie może posiadać tego samego id jak GeoJSON `Feature`. Innymi słowy, `stop_id`, `location_group_id`, i `id` wszystkie stanowią tą samą przestrzeń nazw.

trips.txt (rozszerzenie)

Field Name	Details

`mean_duration_factor`
 	
i
 	
`mean_duration_offset`
	(Float, **Conditionally Required**) Pole `mean_duration_factor` i `mean_duration_offset` umożliwia kalkulacje `MeanTravelDuration` bazując na `DrivingDuration`. `MeanTravelDuration` jest dostarczana przez następującą formułę:
	
	`MeanTravelDuration = mean_duration_factor × DrivingDuration + mean_duration_offset`
	
	`DrivingDuration` jest czasem w którym pojazd przejedzie dystans dla trasy elastycznej (zdefiniowane przez konsumenta danych).
	`MeanTravelDuration` jest obliczoną średnią czasu w jakim oczekuje się przejazdu. Może być obliczone w czasie i dniu trasy. Przewoźnik poruszający się w mieście może mieć factor ustawiony na 1 (bo zawsze porusza się po mieście). Jeżeli chodzi o pojazdy poruszające się pomiędzy miastami faktor powinien być ustawiony na 4 lub 5, ponieważ pojazd często wjeżdża w różne miasta pomiędzy punktami A-do-B, często może również pominąć miasta jadąc drogą szybkiego ruchu.
	
	Na przykład: Faktor ustawiony na 1.5 jest równoważne z powiedzeniem, że “Zwykle, dojeżdżamy do miejsca nie dłużej niż 50% czasu dłużej niż prowadziłbyś sam” lub “To zwykle zabiera 20 minut ten dojazd do tej stacji, gdy obsługuję DRT serwis będę tam do 30 minut, gdyż muszę obskoczyć dwa punkty po drodze”.
	
	Conditionally Required:
	- **Wymagany** kiedy jest przynajmniej jeden stop_time jest zdefiniowany jako grupa lokalizacji.
	- **Zabroniony** w innym wypadku

`safe_duration_factor`
 	
i
 	
`safe_duration_offset`
	(Float, **Conditionally Required**) Pola `safe_duration_factor` i`safe_duration_offset` Pozwalają na przeliczenie `SafeTravelDuration` bazując na `DrivingDuration`. `SafeTravelDuration` jest dostarczany przez następującą formułę:
	
	`SafeTravelDuration = safe_duration_factor × DrivingDuration + safe_duration_offset`
	
	`DrivingDuration` jest czasem w którym pojazd powinien przejechać dany dystans. `SafeTravelDuration` jest kalkulowanym czasem na podstawie 95% ostatnich tras. Przewoźnik może przeliczać ten czas.

	Conditionally Required: - **Wymagany** jeżeli istnieje chociaż jeden stop_time jako grupa lokalizacji - **Zabroniony** w innym wypadku.

stop_times.txt (rozszerzenie)

Field Name	**Details**
`stop_id`	Jeżeli usługa jest na żądanie, lokalizacja GeoJSON lub grupa lokalizacji mogą być referencjami z: - Pola `id` z `locations.geojson` - Pola `location_group_id` z `location_groups.txt`
`to_stop_sequence`	(większe od 0, opcjonalne) - Specyfikuje, że grupa lokalizacji zdefiniowana przez “stop_id” może być osiągnięta w każdym momencie pomiędzy aktualnym “stop_sequence” i “to_stop_sequence”. Może być zdefiniowany dla stop time na grupie lokalizacji. Jeśli zdefiniowane, musi zawierać `stop_sequence`, które jest: - Wyższa wartość niż `stop_sequence` stop time tego przystanku, i - Używana przez stop time danej trasy Przykład: - Bus jeździ przez sekwencję przystanków w specyficznym układzie, jednakże podczas całej trasy, może wykonać odchylenie z trasy aby odwieźć kogoś pod dom np. 100 metrów od wyznaczonej trasy. Region będzie zatem reprezentowany przez grupę lokalizacji (GeoJSON) i stop time będzie posiadał wartość `stop_sequence=0` oraz `to_stop_sequence` z maksymalną wartością `stop_sequence` w tej trasie.
`arrival_time`	([...], **Conditionally Required**) [...] **Conditionally Required**: - **Zakazany** jeżeli `stop_id` wskazuje na GeoJSON lub grupę lokalizacji - [... Dotychczasowa dokumentacja…]
`departure_time`	([...], **Conditionally Required**) [...] **Conditionally Required**: - **Zakazany** jeżeli `stop_id` wskazuje na GeoJSON lub grupę lokalizacji - [... Dotychczasowa dokumentacja…]

`min_arrival_time`
 	
i
 	
`max_departure_`
	(Time, Opcjonalny) Usługa do lokalizacji GeoJSON lub grupy lokalizacji zdarza się w pewnym okresie czasowym, ponieważ dokładna lokalizacja jest nieznana.
	Pola `min_arrival_time` i `max_departure_time` definiują początek i koniec okna czasowego.

`time`	
`pickup_type`	[...] Grupy Lokalizacji: - Nie może posiadać `pickup_type=0`. - Może posiadać `pickup_type=1` lub `pickup_type=2`. - Może posiadać `pickup_type=3` tylko jeżeli jest typem GeoJSON LineString, jeżeli implikuje możliwość komunikacji na poziomie pasażer / przewoźnik.
`drop_off_type`	[...] Grupy lokalizacji: - Nie mogą mieć `drop_off_type=0`. - Mogą mieć `drop_off_type=1`, `drop_off_type=2` lub `drop_off_type=3`.

Konieczne zmiany w bazie danych:

a) Point Deviation - w GTFS opisane jest, że przystanki muszą być dostarczone w odpowiednim ułożeniu (kolumna order), definiowane przez stop_sequence. Propozycja rozszerzenia to dodanie nowego pola “unordered”, które opisuje przystanek powstały przez “przewóz na żądanie” dynamicznie.

Wnioski: Ponieważ jest to otwarto źródłowe opracowanie, które na rok 2020 dalej jest rozwijane, ciężko byłoby zastosować i wdrożyć w projekcie to rozwiązanie. Nie mniej w celach chęci obsługi tras “na żądanie” przy jednoczesnym wykorzystaniu i stosowaniu technologii GTFS, należy się projektowi przyglądać, brać udział w jego rozwoju oraz wyciągać wnioski. W obecnym stanie, technologia byłaby ciężka do zastosowania w takiej formie.

Analiza - Problemy Przewoźnika obsługującego trasy na żądanie

1. W przypadku przejazdów Door 2 Door, na żądanie z obsługą punktów w wyznaczonych miejscach trasy - możliwość sprzedaży biletów on-line jedynie w zakresie pewnego pasma wzdłuż głównych dróg. Inny tryb rodzi problemy.

   Rozwiązanie: Uproszczone narzędzie dodawanie trasy, gdzie przewoźnik rysuje obszar z którego chce odbierać pasażerów.

2. Przewoźnik nie jest w stanie zabrać pasażera, konieczność przepływu informacji oraz przekazanie takiego pasażera innemu przewoźnikowi - za pomocą wyprodukowania jakichś narzędzi. Nie musieliby blokować miejsc specjalnie w wygenerowanych trasach.

   Rozwiązanie: Narzędzie do przekazywania pasażera, w momencie kiedy przewoźnik otrzymuje telefon od pasażera i nie może go zabrać, narzędzie powinno posiadać możliwość wprowadzania takiego pasażera do systemu wymiany pasażerów.

3. Przewoźnicy wykonujący trasy na żądanie mogą dowolnie odmawiać przewozów pasażerom z różnych powodów bez konsekwencji.

   Rozwiązanie: odmowa nie powinna wpływać na skuteczność. Pasażer powinien być przekazany do kolejnych przewoźników - automatycznie bądź manualnie.

4. Problemy przewoźników którzy muszą odmówić pasażerowi: brak miejsc, adresy dostarczania nie są po drodze danej trasy. Nikt bez konkretnej przyczyny nie odmówi zabrania dodatkowego pasażera. Jedyne powody to

brak wyjazdu, brak miejsca lub adresy nie po trasie. Dlatego, jednym z rozwiązań jest danie czasu przewoźnikowi na sprawdzenie "dostępności" i ewentualne potwierdzenie zamówienia lub nie. Tak wszystko działa, dzwoni się do fryzjera - pytamy o wolny termin, zamawiamy miejsce w hotelu - sprawdzamy wolny termin, idziemy do kina - sprawdzamy wolne miejsca itp. itd. Nawet jak klient dzwoni bezpośrednio do przewoźnika to przecież zaczyna się od sprawdzania dostępności miejsc i sprawdzenia adresów. Jeżeli nie, Flotea nie posiada funkcjonalności na potwierdzenie zamówienia przez przewoźnika to najzwyczajniej w świecie trzeba się liczyć z tym, że będą się zdarzały anulowania. Wiadomo, że każdemu zależy na największej skuteczności dlatego klientom powinny wyświetlać się wyłącznie firmy które faktycznie, obsługują dane adresy i dany termin.

Rozwiązanie: Wyszukiwarka przejazdów na bazie obszarów obsługi wprowadzonych przez przewoźnika. Potwierdzanie zamówienia za pomocą dynamicznej aplikacji, która automatycznie pobiera np. Token podczas potwierdzania zabrania.

5. Problem przewoźnika, to pasażer, który jedynie chce się zorientować lub chce znaleźć najtańszego przewoźnika.

   Rozwiązanie: Blokada możliwości zabierania jednego numeru telefonu / pasażera przez wielu przewoźników. Kto pierwszy zabierze pasażera, ten powinien go obsłużyć. W innym wypadku występuje konflikt interesów.

6. Umożliwienie wniesienia opłaty rezerwacyjnej przewoźnikowi za pomocą mechanizmów. Podczas rozmowy przewoźnik musiałby zdecydować czy się umówić z pasażerem czy też nie.

Rozwiązanie: Interakcja zabierania w aplikacji mobilnej, gdzie kliknięcie kosztuje, jest to również informacja dla innych przewoźników, którzy aktualnie obserwują kontakt. Blokada kontaktu na wyłączność podczas kliknięcia.

7. Podczas decyzji przewoźnika należałoby dostarczyć do pasażera a) link do płatności b) dane do przelewu c) inna opcja płatności np. SMS.

   Rozwiązanie: W przypadku kiedy przewoźnik posiada takie ustawienie. Może wysłać standardową zaliczkę automatycznie podczas kliknięcia zabieram. Po kliknięciu zabierz - dwie możliwości. Pierwsza to automatyczne w tle wysłanie linka do płatności stanowiąca standardową zaliczkę, druga to manualne wysłanie za pomocą własnego telefonu linka do płatności (automatycznie wyskakuje okienko SMS). Zaliczka automatycznie byłaby dzielona na np. 50% - gdzie 50% kwoty zaliczki trafia na konto obsługi portalu, a 50% na konto przewoźnika w formie punktów / Tokenów. W przypadku automatycznego wysyłania sms w tle - należy doliczyć opłatę pobrania kwoty z Tokenów przewoźnika za wysłanego SMS.

8. Niektóre problemy można rozwiązać decyzją o tym czy przewoźnik dogadał się z pasażerem powinna należeć do przewoźnika. On powinien chcieć "zarezerwować" sobie ten kontakt.

   Rozwiązanie: Kliknięcie w “Zabieram” po lub w trakcie rozmowy z pasażerem w aplikacji mobilnej lub w aplikacji webowej, powoduje blokadę kontaktu dla innych przewoźników.

9. Przewoźnik dostaje sms z danymi pasażera i od razu moze dzwonic. W tym SMSie podobnie informacja o tym żeby odpisał TAK jeśli - go zabiera. I tu mamy super funkcje - czyli rozsyłamy do 3 przewoźników, jeśli żaden w ciągu

15 minut nie odpisze "tak" rozsyłamy do następnych 3 przewoźników. ;]. Jeśli jeden z tych 3 odpisze "tak". Powinniśmy resztę przewoźników poinformować o tym, że pasażer został już zabrany przez innego przewoźnika - żeby nie dzwonił do niego niepotrzebnie. Resztę - czyli 2 przewoźników z 3 rozesłanych w tej iteracji.

Rozwiązanie: Grupowanie przewoźników w zależności od np. za pomocą systemu oznaczeń np. Medali. Kontakt do pasażera powinien być przekazywany do następnej grupy przewoźników po upływie czasu.

10. Przykład opłaty rezerwacyjnej - w momencie kiedy wysyłane jest informacja “TAK” przewoźnik kliknie w aplikacji mobilnej, że go zabiera itp. To uruchamia się proces: wymuszenia wpłacenia opłaty rezerwacyjnej - I tutaj zadanie Flotea / Call center / czy automatyczne. Opłata rezerwacyjna w całości zostaje we Flotea - ale przewoźnik ma pewność, że pasażer pojedzie. Jeśli opłata rezerwacyjna zostanie wniesiona - to super Flotea zarabia np 30 zł, a to jest budżet do obsługi tego pasażera - czyli Flotea dzwoni, upewnia się dodatkowo czy jedzie, informuje przewoźnika o tym pasażerze. Obsługuje jedna i druga stronę, aż do momentu wyjazdu (opieka nad pasażerem). Jeśli pasażer nie wpłaci, to nie ma 100% pewności że pojedzie. Czyli Wszystko zależy od tego - żeby przewoźnik poinformował o tym, że go zabiera - wiedząc, że w ten sposób upewnij się, że pasażer pojedzie.

    Rozwiązanie: Link z opłatą rezerwacyjną jednakże, nie w całości dla portalu. Można tutaj założyć, że jeżeli opłata rezerwacyjna wynosi tylko 30 zł, wtedy stosuje się formę 50/50, wtedy do przewoźnika zostaną dodane Tokeny o wartości 15 zł. W przypadku jeżeli przewoźnik pobierze zaliczkę dużo większą np. Wynoszącą całość kwoty biletu - wtedy maksymalna prowizja będzie wynosiła 30 zł na korzyść portalu,

podczas gdy konto przewoźnika zostanie doładowane pozostałą kwotą np 270 zł przy kwocie 300 zł.

Wnioski:

Rozwiązanie to umożliwia opracowanie nowego typu kontraktów dla celów transportowych, służących bardziej w celach rozwiązania problemów specyficznej grupy przewoźników - tych, którzy zajmują się przejazdami i generują trasy na żądanie. Przewoźnik tego typu nie wytwarza statycznych tras, raczej coś w rodzaju zbierania zgłoszeń, z których następnie budowane są trasy. Często w tym wypadku nie ma miejsca i czasu na optymalizację. Trasy są wyznaczane przez przewoźników w dowolnym przez siebie momencie. Np. podczas ręcznego pakowania pasażerów do pojazdów i wskazywania tras. Na ten moment (08.2019) Nie znaleziono na rynku narzędzi, które umożliwiałyby optymalizację pracy tych przewoźników.

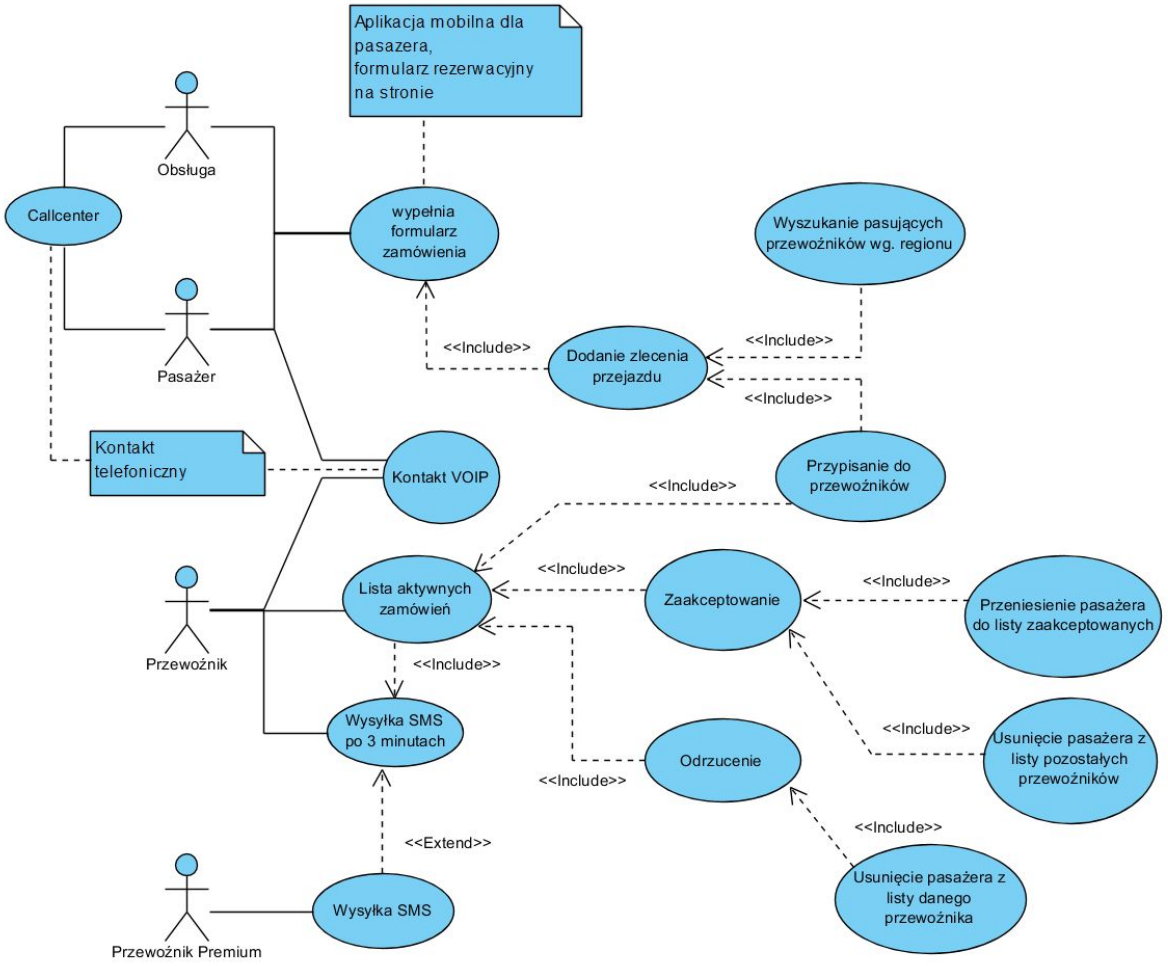
Wymagania:

- Przewoźnik otrzymuje ofertę, gdzie kontakt kosztuje 2 Tokeny - wdzwonienie się do pasażera.
- Przewoźnik reaguje na ofertę np. Klikając przycisk “zabieram”, wtedy pobierane jest np. 10 Tokenów FLT
- Generowanie zaliczek, które w momencie opłacenia przez przewoźnika, rozdzielane są pomiędzy kontrakt, a przewoźnika (np. Określona wartość do pewnej sumy). W przypadku kwot mniejszych niż 100 zł, stosowanie zasady 50/50. W przypadku kwot większych niż 100 zł, stosowanie zasady 10%. Wtedy za pomocą Tokenów kontrakt automatycznie odliczały kwoty z konta.

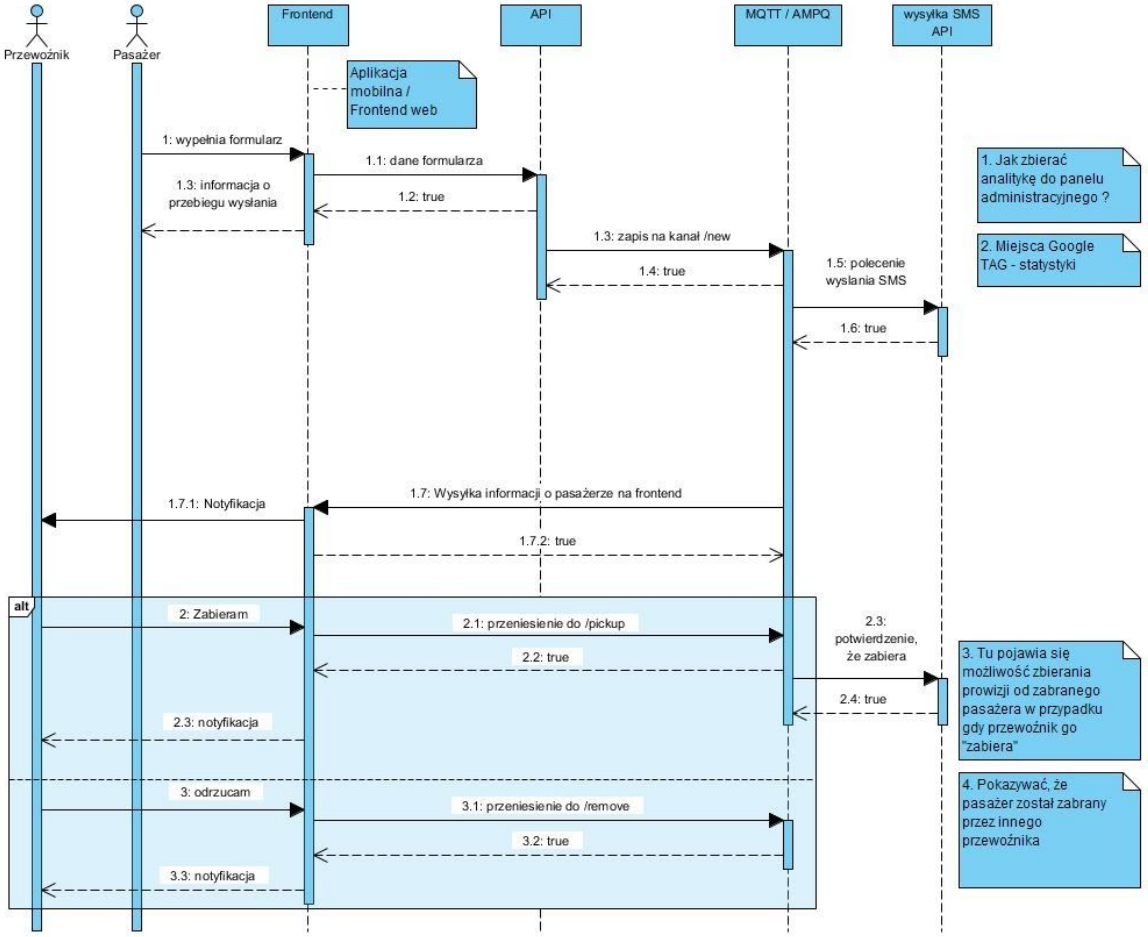
Zalecenia:

- Wykorzystanie wielu technologii komunikacyjnych takich jak MQTT
- Zbudowanie prototypu narzędzia do zamawiania przejazdów door 2 door

- Zbudowanie backendu i systemu kredytów, który będzie działał w oparciu o technologię Blockchain.
- Zbadać możliwości pobierania płatności z innych miejsc



Powyższy schemat prezentuje w jaki sposób wyglądają mniej więcej przypadki użycia i proces zamawiania przejazdu przez Pasażera. Poniższy diagram natomiast przedstawia proces zabierania pasażera wykorzystując technologię MQTT.



Wnioski: Aby usprawnić działanie przewoźników oraz usprawnić działanie procesów zamawiania przejazdów należy zbudować prototyp aplikacji mobilnej do zamawiania przejazdów oraz aplikację do obsługi zamówień dla przewoźnika. Projekt prototypu aplikacji - można przygotować w taki sposób, aby umożliwić stosowanie tokenów FLT w przyszłości (tak aby dało się do technologii zastosować rozwiązania). Zaleca się zatem wszystkie funkcje, które dotyczą operacji na kredytach przenieść na funkcje pisane po stronie bazy danych.

PRZYDATNE LINKI

1. http://postgis.net/source - obiekty dla PostgresSQL pozwalające na wyszukiwanie pozycji
2. http://www.w3.org/TR/poi-core/ - opisy punktów zainteresowania POI W3C
3. https://en.wikipedia.org/wiki/General_Transit_Feed_Specification
4. OpenTripPlanner - przykład plannera OTP opartego o GTFS http://docs.opentripplanner.org/en/latest/
5. https://docs.google.com/document/d/1QVrijt2omNN_IvCkC0FmRhciVmWg-cJtNW_PqULc-Pw - Publiczny dokument nowego standardu GTFS Flexible, tworzony do tej pory.
6. https://github.com/OpenTransitTools/gtfsdb - narzędzia OTT, przykład implementacji bazy danych GTFS
7. https://transitfeeds.com/l/434-poland - baza feed transitu, która może posłużyć do testowania
8. https://github.com/MobilityData/gtfs-best-practices/blob/master/en/about.md - dobre praktyki przy stosowaniu standardu GTFS