Badania i Analizy nad momentami dokonywania transakcji i warunkami kontraktu

Projekt: “Przeprowadzenie prac badawczych i rozwojowych umożliwiających wdrożenie inteligentnego kontraktu opartego o technologię blockchain”

Momenty dokonywania transakcji i warunki kontraktu jakie muszą zostać spełnione aby obsłużyć wybrane procesy transportu osobowego.

Autorzy:
- Blicharski Bartłomiej
- Martin Morawiec
- Tomasz Sienkowiec

Kontekst: Badanie będzie skupiało się na potrzebach błyskawicznego realizowania i rozliczania połączeń przewoźników umieszczonych w sieci Blockchain. Zespół badawczy skupi się na tym aby wyodrębnić najlepsze sposoby opracowania Inteligentnych Kontraktów umożliwiających obsługę procesów istniejących w gałęzi transportu osobowego.

Opis ogólny z projektu:

Opracowanie Inteligentnego kontraktu obsługującego transport zbiorowy.

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

I w ramach Inteligentnego kontraktu (IK) na usługi transportowe:

1. **Aplikowaniu połączeń** - główna metoda kontraktu dodanie połączenia aktywnego w danym momencie
2. **Konfiguracja miejsc / ilości miejsc**
3. Oferowanie różnych typów środków transportu np.: przejazd busem, pociągiem itp.
4. **System rozliczeń i transakcji** – możliwość zapłaty za kontrakt kryptowalutą, aż do wypełnienia miejsc dostępnych w ramach wytworzonego kontraktu.
5. **Błyskawiczna**, **natychmiastowa płatność** - możliwość pośredników finansowych takich jak banki
6. **Natychmiastowe rozdzielenie płatności** pomiędzy uczestników kontraktu np.: przewoźnik, agent, twórca kontraktu

II w ramach modułu komunikacyjnego API

1. **Interfejs obsługi IK:** dodawanie połączeń, określanie typu połączenia, konfiguracja agenta itp.
2. **Interfejs pobierania informacji z IK o połączeniu:** liczba sprzedanych biletów, kwoty zebrane przez IK, typ kontraktu itp.
3. **Interfejs pobierania informacji z Inteligentnego Kontraktu o transakcjach Przewoźnika i Agenta**

Spotkania

a) Spotkanie I - z dn. 17.04.2018

Uczestnicy: Bartłomiej Blicharski, Tomasz Sienkowiec, Martin Morawiec
Miejsce: Biuro nr 11, Cieszyn

Występujące Role w kontrakcie:

a. Flotea
b. Przewoźnik
c. Pasażer
d. Agencja

Występujące czynności w projekcie:

- Przewoźnik tworzy trasę która powinna być wrzucona do publicznej bazy danych - automatycznie, która działa w ramach kontraktu Flotea
- Pasażer kupując bilet przystępuje do kontraktu w którym uczestniczy a i b
- Agencja kupując bilet tak samo jak pasażer przystępuje do kontraktu z a i b
- Pasażer kupując bilet poprzez portal agencji przystępuje do kontraktu w którym uczestniczą a, b, d
- Zastanawiamy się nad tym, aby nikt nie widział że ma portfel w portalu Flotea lub w portalu w którym kontrakty będą używane, tylko żeby rozliczenie było robione na walucie zakupu biletu.

Realizacja kontraktu - rozpoczęcie i zamknięcie przy zakupie pasażera

- Od strony portalu, rozpoczęcie odbywałoby się po kliknięciu KUP BILET lub w momencie rozpoczęcia trasy.
- Zamknięcie automatyczne przy wysiadaniu z Autobusu - ale tutaj potrzebujemy danych lokalizacji urządzenia mobilnego pasażera.
- Ewentualnie lokalizacja aplikacji u kierowcy w pojeździe - podobnie jak w Uberze, rozpoznanie urządzenia pasażera przy zbliżeniu się do urządzenia kierowcy na x odległość - to byłoby fajne dla taxi i przewozach door to door. Wtedy pasażer nie musiałby wyszukiwać oferty, tylko oferta pojawiała by mu się na aplikacji i w momencie ruszenia auta otwarty zostałby kontrakt.

Obsługa Last Minute i First minute - przykładowo 10% wyjazdów będzie sprzedawane po 30% niżej. Kolejne sprzedaże od 10% do 80% po cenie normalnej. A od 80% - 20% jest sprzedawane po cenie last minute. Modulacja ceny w zależności od natężenia sprzedaży? Last minute - jeśli ktoś zrezygnuje z biletu i stracił np 80% to mozna uwolnic ten kontrakt jako last minute i za kwotę dopełnienia np. 20% ceny (wtedy osoba kupująca last minute płaci jedynie 20%). Określenie od kiedy do kiedy ma być możliwe oddanie bez kosztów biletu (określenie przez przewoźnika - czy w jaki sposób?)

- Czy możliwe jest sprzedawanie własnego biletu / zmiana właściciela kontraktu ?
- Oparcie kontraktu na kalendarzu ? Data wyjazdu / przyjazdu.
- Bilet Open ? Data ważności (od kiedy - na ile czasu). Pojawia się problem zamykania kontraktu

- Bilet dwustronny ? Kontrakt powrotu - jedziesz jutro, ale nie wiesz kiedy wracasz do warszawy - zakup kontraktów/powrotów za niższą cenę bez określonej daty powrotu.

Zamykanie kontraktu:

- moment kasowania biletu,
- albo określona data (przy dodawaniu kontraktu lub przez pasażera),
- albo zwrot kontraktu.

Temat blokowania miejsc w kontrakcie:

- Czy może istnieć grupa kontraktów nadzorowana przez kontrakt nadrzędny? Chodzi o blokowanie miejsc... (grupa kontraktów - instancje, odcinki obsługiwane na trasie). Czy jeżeli ktoś zakupi bilet w sub-kontrakcie to czy możliwe jest przeliczenie wolnych miejsc w pozostałych kontraktach ?
- Zastanowić się czy to ma sens w przypadku innych przejazdów niż autokarami / busami.
- Kontrakty door 2 door.

Martin zastanawiał się nad tym czy:

- Kontrakt z portfelem, który ma zniżki
- Wykorzystanie kontraktów, które się odbyły do statystyk przewoźników, tras i użytkowników
- Czasowy / kilometrowy bilet - użytkownik wpłaca do kontraktu krypto na używanie takiego biletu. Kontrakt liczy kilometry (tutaj nie wiem jak) lub czas (tutaj nie wiemy jak) w busie (lub inny sposób rozliczenia)
- System cenowy

Bartek zastanawiał się nad:

- Uproszczeniem kontraktów transportowym, w taki sposób aby kontrakt zawierał jedynie takie dane, które są potrzebne do jego zrealizowania
- Rozszerzeniem zakresu działania o wszystkie możliwe typy transportu z punktu A do punktu B

- Zniżki są istotne, trzeba znaleźć na nie rozwiązanie
- Kontrakty z opcją first i last minute oraz komplikacjami związanymi z. First i Last minute mają określone działania. Np. Last minute działa, jeżeli bilet x został zwrócony - wtedy miejsce się zwalnia.. Właśnie wtedy pojawia się last minute. Jeżeli rozwiązanie będzie nosiło znamiona trudności i mocno będą komplikowały sprawę należy oddalić się od ich realizacji.
- Kontrakt powinien zawierać dokładne dane geograficzne z punktu A do punktu B (długości i szerokości geograficzne), typ pojazdu, ilość miejsc. Powinien zawierać typ przejazdu itp. Powinien być wykonany w taki sposób, aby był jak najbardziej optymalny i prosty, zamiast skomplikowany, ciężki i trudny. Dotychczasowe prace nad portalem pokazały, że dokładanie danych, komplikowanie przejazdów i utrudnianie funkcjonalne, tak naprawdę nie wpływa pozytywnie na popularność, a bardzo negatywnie na wydajność systemu (nawet nieużywanego).
- Należy dokładnie się zastanowić i pisać kontrakty z użyciem blockchaina, który umożliwi jego łatwą skalowalność (analiza pod tym kątem).

b) Spotkanie II - Burza mózgów z dn. 10.05.2018

Uczestnicy: Bartłomiej Blicharski, Tomasz Sienkowiec, Martin Morawiec
Miejsce: Biuro nr 11, Cieszyn

Podczas rozmowy chcieliśmy nakreślić w jaki sposób i czy warto dodawać punkty “przystanków” do kontraktu. Zaproponowany sposób wygląda następująco:

1. Opcjonalna tablica z punktami przejazdów / przystanków (opcjonalne)

   Aby zapewnić funkcjonalność “przystanków” w “połączeniu” należy np. Dodać punkty przejazdu do tablicy w kontrakcie (z pewnym ograniczeniem).

   Np. stops: [A:{lat: 1.000000, lng: 1.000000},B:{lat: 1.200000, lng: 1.200000}]

   - Należy się zastanowić czy coś więcej, zaproponować jakiś kontrakt.

2. Tablica wycen przejazdu (opcjonalna)

   Aby zapewnić “różnicę” w dowozach do poszczególnych punktów

   [A: {B:100;C:100]] - Oznacza, że z przystanku A można dojechać do przystanku B i C w cenie po 100 jednostek;
   [B:{A:100;B:100;C:100}] - BŁĘDNE. Oznacza, że można dojechać z przystanku B do A (co jest błędne) i z B do B (co jest błędne) i z B do C (poprawne) w kwocie 100 jednostek;

3. Pojazd ? Miejsca w kontrakcie wg. ID. Z uwzględnieniem, że konkretna osoba może zakupić konkretne miejsce. (Opcjonalne)

   Seats: [1:0,2:0,3:1,4:1] - Oznacza, że w pojeździe są 4 miejsca w tym jedno zajęte z nr. 3. Przy zakupie opcjonalnie trzeba wskazać: które miejsce, oraz którego odcinka dotyczy.

- Problematyka związana z blokowaniem miejsca w pojeździe na dany odcinek, nie może być absolutnie skomplikowana. Ew. przełożyć na portale / SDK.
- Miejsca komplikują się z odcinkami. Odcinki są wykonywane w różnym czasie. Więc w teorii trzeba myśleć również o jakimś timestampie, który oznaczał będzie odjazd z danego przystanku.

4. Zakup Powrotu? Bilet Open ? - Trzeba się zastanowić nad komplikacją tych rozwiązań. Jak bilet Open przypisywać do danego miejsca. Osoba z biletem Open może w teorii wejść do każdego pojazdu na danym odcinku, nawet jeżeli wszystkie miejsca są zajęte. Nie może tak być. Zastanowić się nad sensownością biletów Open.

Pytania

Przechowywanie długości tras (czy jest to obecnie zapisywane? czy powinno być na blockchainie?)

Przechowywanie długości tras, jest ciekawą perspektywa. Myślę, że można byłoby te dane przechowywać. Na pewno w kontrakcie powinny znaleźć się punkty geograficzne (minimum pkt. A i pkt B docelowy w postaci długości i szerokości geograficznej). Z tych danych tych można już po stronie aplikacji frontendu przeliczyć długości tras. Z drugiej strony mając długość już wyliczoną - można byłoby sortować. analizowano inne rzeczami typu: W jaki sposób utrzymywać informacje nt. innych elementów trasy np. przystanki pomiędzy A i B, analizujemy po wdrożeniu jak to zrobił “Google Transit” i pewnie na tym będziemy się wzorować jakoś. We Flotea jest to źle zrobione - każdy wyjazd generuje osobne połączenia z tymi danymi np. pkt A-B, B-C, A-C, A-D, B-D itp. generuje strasznie dużo ofert - niestety nie jest to optymalne rozwiązanie, a tym bardziej nie wyobrażam sobie jak to “dodać” wszystko do blockchaina (lepiej to zrobić w ramach 1 kontraktu danej trasy - tutaj właśnie google i scheduler wzorujący się na POI DataProvider technologii FI-WARE, są jakimiś punktami zaczepienia dla nas.

Obecny sposób wyliczania ceny biletu (kilometry/różne ceny dla odcinków)

Kilometry nie są do zrealizowania (to jest popularne w branży TLS w transporcie Towarów). W transporcie osobowym płaci się za przejechany odcinek. Więc raczej będziemy stosowali wyceny na odcinki / całą trasę (nie wiadomo jeszcze jak. We Flotea wycenia się odcinki (podczas tworzenia na mapach trasy). Przewoźnicy wyceniają sobie poszczególne odcinki np od pkt A-B, B-C itp. W sumie bardziej zależy od typu transportu np. autobus międzymiastowy, który sprzedaje bilety pomiędzy pobliskimi miastami jeździ tak że sprzedaje bilety na każdy odcinek. Nie wiem czy chcemy to aż tak komplikować.

Wpadliśmy na pomysł żeby w jednym kontrakcie trzymać info (tak jak robi to Google Transit) nt. Punktów przez który przejedzie pojazd Czyli dwa główne A - D (początek i koniec trasy), a w kontrakcie trzymać info jeszcze o tym ze sa pkt B i C. Ewentualnie myślę, że można zrobić opcjonalną metodę do uzupełniania wycen na poszczególne połączenia pomiędzy punktami w formie jakiegoś jsona czy tablicy np.

setPrices({A:{B:10,C:10},B:{C:5}}) co oznaczałoby, że cena z pkt A do B wynosi 10, z A do C wynosi 10, z B do C wynosi 5. Problemem jest też blokowanie tego samego miejsca przez osoby jadące na różnych odcinkach np. A-B i B-C nie blokują się wzajemnie. Jeśli miejsce A-C jest wykupione to już B-D nie może zostać wykupione. Tutaj pewnie kwestia jakiegoś dobrego algo, który to sprawdza.

Ograniczenia przy dodawaniu trasy

Przewoźnicy nie powinni oferować dokładnie takiego samego przejazdu (gdy ta sama trasa i czas) Ograniczenie to bardzo ważny temat. Pytanie tylko czy takie sprawdzanie jest możliwe bez użycia gazu np. zastanawiamy się nad zrobieniem tego pod casper / sidechain, aby zoptymalizować działanie.

Scenariusze

Scenariusz: Wyszukanie połączenia zapisanego w sieci blockchain

1. Użytkownik wyszukuje połączenia (punkt startowy, punkt końcowy, data)
2. Użytkownik dostaje propozycje różnych połączeń z tego dnia oferowanych przez różnych przewoźników

Scenariusz ten jest dobrze rozumiany. Zastanawiam się ostatnio mocno jak to zrobić. Samo wyszukiwanie bezpośrednio na blockchainie kontraktem - niestety nie widziałem jeszcze takiej możliwości. Na pewno da się w prosty sposób wyciągać dane, ale skomplikowane zapytania odpadają. Dlatego zastanawiałem się nad tym, żeby w kontraktach zostawiać podstawowe “konieczne do wykonania kontraktu dane”.

Potencjalne rozwiązania (we wdrożeniu do portali):

1. SDK z wyszukiwarką, która operuje na umieszczonym tam gdzie wyszukiwarka kopii blockchaina. Chodzi o to, żeby wykonać kod łączący świat blockchaina z portalami. Portale wtedy mogą użyć SDK u siebie i będą mieli do dyspozycji “wyszukiwarkę” z API. SDK to automatycznie zaciągnie np. blockchaina i go uruchomi, dzięki temu wyszukiwarka uruchomiona na portalu X oprócz dostarczenia API, dostarczy także “bazę”.

2. Zastanawiałem się nad funkcjonalnością “śledzenia bloków” z transakcjami w kontekście kontraktów. Narzędzia wtedy, mogłyby aktualizować np. jakąś lokalną bazę dostarczaną wraz z narzędziami np. postgresql/mongodb, dodając do niej połączenia i po niej wyszukiwać. Opierając się o FI-WARE.

Scenariusz: Zakup biletu bez profilu (bez blockchain)

1. Po wybraniu połączenia
2. Użytkownik decyduje się na zakup

Flotea - obecne działanie:

- Pasażer znalazł w wyszukiwarce połączenie, ma pobrane informacje o kontrakcie na to połączenie (wyniki są pobierane z lokalnej bazy Flotea)
- Pasażer wybiera ofertę (wypełnia formularz zamówienia)
- Wykonuje
    - płatność online - za pośrednictwem payu kwota przekazana jest do portalu flotea, na konto bankowe (1.9% prowizji), następnie bank również pobiera prowizję. Kolejnym krokiem w interwałach 2 tygodniowych lub miesięcznych, kwoty są przelewane do przewoźnika. We Flotea zostaje prowizja.

    - rezerwacje - uruchamia proces komunikacji mailowej przewoźnik / pasażer. W tym wypadku strony dogadują się telefonicznie.

Kontrakt:

- Pasażer znalazł w wyszukiwarce połączenie, ma pobrane informacje o kontrakcie na to połaczenie
- ~~Miejsce jest rezerwowane~~
- ~~Przewoźnik dostaje informacje o rezerwacji~~
- Użytkownik wpłaca pieniądze.
    - Token zostaje wpłacony do kontraktu Flotea (który może być również kontraktem).
- Flotea, Agent i Przewoźnik otrzymuje błyskawicznie token
    - Kontrakt powinien być czasowy, w momencie uruchomienia połączenia (wyjazd, rozpoczęcie podróży), kontrakt jest uruchamiany (tokeny są wysyłane na konto kontraktu (jakiś 0.5-1%), na konto Agencji (% zależny od ustawień np 10%), na konto Przewoźnika (w tym wypadku jakieś 89%). Jeśli przewoźnik jest jednocześnie agentem - to się sumuje (sytuacja kiedy przewoźnik sam umieszcza kontrakty i sam je sprzedaje).
- ~~Później pieniądze są przekazywane do przewoźnika~~

Scenariusz: Założenie profilu z opcją płatności krypto

1. Użytkownik zakłada konto w serwisie
2. Użytkownik dostaje token czasowy (nie mylić z tokenem sieci blockchain)
3. Token w serwisie jest powiązany z użytkownikiem
4. Użytkownik loguje się na MetaMask
5. Użytkownik wpłaca środki do kontraktu przekazując token id
6. W kontrakcie zostaje powiązany adres portfela z tokenem
7. Przez oracle/event kontrakt sprawdza czy token został wygenerowany w portalu i jest ważny (?)
8. Jeśli token jest ważny to zostaje unieważniony i klucz publiczny zostaje powiązany z id.

Scenariusz: Próba oszustwa

1. Oszust generuje losowy token i przelewa pieniądze
2. Token zostaje powiązany z portfelem
3. Token nie istnieje w serwisie więc użytkownik pozostaje anonimowy

Uwaga: Niewielkie zagrożenie bo wymaga przelania środków

Scenariusz: Płatność w kryptowalucie

1. Użytkownik ma profil powiązany z portfelem
2. Użytkownik decyduje się na zakup biletu
3. Użytkownik loguje się do MetaMask i potwierdza transakcję / lub za pomocą innego sposobu
4. Użytkownik otrzymuje bilet - numer (QR kod)

Scenariusz: Wygenerowanie biletu

1. Po zapłacie użytkownik otrzymuje numer (QR na telefon)
2. Numer biletu jest zapisany w kontrakcie

Miejsce: wdrożenie

Jak to się odbywa we Flotea: po opłaceniu transakcji (kilka - kilkanaście minut potwierdzenia z payu, w zależności od tego z którego banku przenika płatność) status biletu zmienia się na opłacony. Wtedy pasażer oprócz samego dostępu do biletu online (lista biletów we Flotea) otrzymuje również na e-maila sam bilet w postaci PDF. Przewoźnik otrzymuje informacje o tym, że bilet został sprzedany. Bilet jest też dostępny w aplikacji mobilnej dla pasażera.

W przypadku Kontraktu, wystarczy identyfikator konkretnego kontraktu - lub konkretnej transakcji. Portale Frontendowe w tym SDK powinny sobie poradzić z tym. Np. narzędzia mogą mieć konkretne API które będzie pobierało “bilet” wg. Danego identyfikatora. Jest to praca nad którą trzeba popracować obecnie.

Scenariusz: Sprawdzenie biletu I

1. Użytkownik pokazuje otrzymany numer (QR kod)
2. Czytnik sprawdza liczby kontrolne i bieżący przedział czasowy oraz odhacza użyty

kod w wewnętrznej bazie

Czytniki nie są konieczne, różnie jest to realizowane, część przewoźników wie po prostu, że dane miejsce jest zarezerwowane i nie sprawdza nawet osoby siedzącej na tym miejscu. Jeżeli chodzi o aplikację mobilną - wystarczy, żeby kierowca miał aplikację mobilną i będzie mógł porównać bilety, posiada listę identyfikatorów / kontraktów które mogą udać się w podróż. Aplikacja mogłaby wtedy aktualizować statusy biletów interwałowo odpytać blockchain, czy transakcja się odbyła (w przypadku kiedy ktoś kupuje zaraz przed wyjazdem bilet). Przy takim podejściu urządzenie mobilne kierowcy zastępuje cały czytnik (mamy również dane GPSa przez mobile)

Uwaga: Czytnik działa inaczej niż smart kontrakt na blockchain. W smart kontrakcie kod jest widoczny i nie da się ukryć metody sprawdzania sumy kontrolnej

TODO: Czy na blockchainie da się w ogóle generować bilety?

Możemy potraktować uruchomienie kontraktu przez strony jako właśnie bilet - jest to wtedy Smart Bilet, który jest kontraktem. Bilet jest publiczny - strony mogą być anonimowe.

Zagrożenie: każdy może śledzić przejazdy, transakcje są publiczne ? Jak to rozwiązać ? Przewoźnik może nie życzyć sobie tego, żeby udostępniać jego dane “transakcyjne”. Ktoś może śledzić sprzedaż biletów konkurencyjnej ceny. Można “dowiadywać” się i zbierać dane statystyczne nt. przejazdów z wybranych miejsc “publicznie”. Strony, przewoźnicy itp. powinny być anonimowe w sieci blockchain, bez szczegółowych danych.

Scenariusz: Sprawdzenie biletu 2

1. Kierowca przed wyjazdem otrzymał listę biletów na cały przejazd
2. Pasażer pokazuje numer przed wejściem do pojazdu (kod QR)
3. Kierowca wyszukuje numer na liście (tzn. system czytnika wyszukuje)
4. Numer jest zaznaczony na liście jako wykorzystany

Scenariusz: Nowe połączenie

1. Przewoźnik dodaje połączenie (punkt startowy, punkt końcowy, czas odjazdu, dzień tygodnia, czas przyjazdu (godzina, dzień tygodnia), liczba miejsc, ceny biletu)
2. Przewoźnik wybiera przedział czasowy obowiązywania połączenia
3. Połączenia są generowane w smart kontrakcie

Uwaga: Czy to smart kontrakt tworzy odcinki i przejazdy w zakresie dat?

Scenariusz: Konkurs - gamifikacja

1. Flotea tworzy nagrody w postaci niewymienialnych tokenów (ERC721 lub ERC1155) na wzór Cryptokitties
2. Nagrody są przypisane do danych tras (danych miejsc na trasie? - Pokemon Go)
3. W trakcie przejazdu wybrani pasażerowie (np. 3 osoby, które pierwsze dokonały rezerwacji) zbierają nagrody
4. Nagrody mogą być wymieniane między użytkownikami
5. Nagrody mogą być odsprzedawane na giełdzie

Scenariusz: Punkty lojalnościowe

1. Flotea w porozumieniu z przewoźnikami ustaliła nagrody/zniżki za zebrane punkty (punkty = wymienialne tokeny na blockchain)
2. Zarejestrowani użytkownicy mają prowadzony rejestr punktów
3. Za każdy przejechany kilometr użytkownik otrzymuje punkt
4. Po zebraniu odpowiedniej liczby punktów użytkownik może wymienić je na nagrody/zniżki/darmowe przejazdy

Dodatkowe inspiracje

Użycie blockchain w transporcie towarowym

Problemy:

- każdego dnia w przemyśle transportowym jest zamrożonych 140 miliardów dolarów w związku z zaległościami w płatnościach
- średni czas oczekiwania na płatność wynosi 42 dni od wystawienia faktury
- koszty administracyjne stanowią już prawie 20% całego kosztu usługi transportowej
- 8,5% przesyłek farmaceutycznych wrażliwych na wahania temperatury nie przechodzi kontroli celnej ze względu na przekroczoną temperaturę
- 90% firm transportowych ma 6 ciężarówek lub mniej. Utrudnia to powiązanie popytu i podaży. Ogromna liczba przewozów ma niepełny ładunek.
- przeciętny mrożony ładunek musi przejść przez około 30 organizacji. Problemy w komunikacji mogą spowodować przetrzymanie ładunku (utratę).

Rozwiązania z wykorzystaniem blockchain:

- rozproszony rejestr pozwala wyeliminować dokumenty transportowe (np. cmr) - eliminacja papierowych dokumentów
- utrzymanie temperatury transportu - IoT i dane na blockchain (monitorowanie parametrów)
- smart kontrakty mogłyby usprawnić proces odprawy celnej (akceptacji towaru)
- usunięcie papierologii
- wiarygodne i nieusuwalne komentarze (podobnie jak w Blablacar)
- zbieranie danych i wymiana wiarygodnych danych pomiędzy firmami w całym ekosystemie
- rejestrowanie historii części wymiennych do ciężarówek
- skalowalny system trackowania

źródło: https://www.winnesota.com/blockchain

Przykład analogiczny do GTFS - Inspiracja Google Transit

Google udostępnia interfejs pozwalający na publikowanie (plikami) danych dotyczących tras. Na pewno będziemy chcieli umożliwić korzystanie z Google Transit przez SDK. Kanał GTFS składa się z serii plików tekstowych zebranych w pliku ZIP. Każdy plik modeluje określony aspekt informacji tranzytowych: przystanki, trasy, wycieczki i inne dane harmonogramu. Szczegóły każdego pliku są zdefiniowane w pliku referencyjnym GTFS.
źródło: https://developers.google.com/transit/gtfs/guides/tools

Sposoby promocji kontraktu w sieci Ethereum

W sieci Ethereum istnieją kontrakty, tworząc kontrakt mamy możliwość wygenerowania tokenów, które zasilą adres właściciela / twórcy. Tokeny te również mogą stanowić świetną formę reklamy.

Tworząc kontrakt w sieci Ethereum otrzymujesz:

- Możliwość tworzenia tokenów
- Pewną ilość tokenów
- Możliwość przesyłania tokenów

Sposobem łatwej prezentacji lub poinformowania użytkownika sieci Ethereum o “marce” tokenu jest możliwość przesłania go na konto adresów które znamy. Posiadamy możliwość zebrania np. Kilkuset wybranych / wszystkich kont w sieci i przesłać tokeny naszego kontraktu na te konta - płacąc odpowiednią ilość gazu. Jest to koszt, którego poniesienie jest równoznaczne z zapłatą za ten sposób docierania do klienta. W odróżnieniu od popularnych usług reklamowych typu Google AdWords system jest zdecentralizowany i nie należy pod żadną konkretną firmę (za wyjątkiem społeczności Ethereum). W popularnych serwisach i portfelu Ethereum np. Mist od tego momentu widoczny jest przypisany Token w pewnej ilości, który będzie istniał dopóki właściciel portfela go nie “wyda”. Kolejnym kosztem, jest sama wartość tokenu w przyszłość. Rozdawanie tokenów można powtarzać w nawet, gdy token jest już coś wart (wtedy koszt reklamy rośnie, ale nie zarabia firma pośrednicząca, a jedynie właściciel kontraktu).

Charakterystyka tokenów i eksperymenty z programowaniem Ethereum

W sieci Blockchain Ethereum, wirtualny token jest używany np. W Inteligentnych Kontraktach, które są używane w kontrakcie lub grupie kontraktów tworzących pewnego rodzaju zdecentralizowaną aplikację. Aplikacje w oparciu o kontrakty można rozumieć jako "żyjące" procesy, które działają dopóki aplikacja jest używana. Napędem działania aplikacji jest gaz.

```
pragma solidity ^0.4.13;
```

contract owned - kontrakt który przypisuje adres, z którego jest wrzucany jako właściciel. Kontrakt zawiera modyfikator onlyOwner, który pozwala na użycie funkcji tylko przez właściciela (za "_;" w modifier wstawiane jest ciało funkcji, która jest wywoływana) i funkcję transferOwnership, która pozwala przypisać nowy adres jako właściciela kontraktu.

```
contract owned {
    address public owner;

    function owned() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require (msg.sender == owner);
        _;
    }

    function transferOwnership(address newOwner) onlyOwner {
        owner = newOwner;
    }
}
```

contract tokenRecipient - kontrakt używany przy dysponowaniu czyimiś tokenami.

```
contract tokenRecipient {
     function receiveApproval(address _from, uint256 _value, address _token,
bytes _extraData);
}
```

contract token - dziedziczy po owned.

Zmienne:

a) name - nazwa tokenu (np. Bitcoin). Ustawiane w konstruktorze (tokenName)
b) totalSupply - ilość wszystkich tokenów. Na początku ustawia się w konstruktorze (initialSupply), przy tworzeniu nowych tokenów wartość ta jest zwiększana symbol - symbol tokena (np. BTC, ETH), może mieć dowolną długość. Ustawia się w konstruktorze
c) decimals - ilość miejsc (np. Ethereum ma 18 -> 1ETH = 10^18 WEI, WEI to najmniejsza jednostka jaką można wysłać). Ustawia się w konstruktorze (decimalUnits)
d) buyPrice, sellPrice - cena zakupu i sprzedaży tokena, właściciel ustawia je w funkcji
e) setPrices(sellPrice, buyPrice)
f) balanceOf - tablica zawierająca ilość tokenów danej osoby (adresu)
g) allowance - tablica zawierająca ilość tokenów jaką może wysłać upoważniona osoba z innego adresu. Przykład: Adres A ma 100 tokenów i pozwala adresowi B wysłać do kogoś 10 z nich. W tym wypadku na koncie adresu A są dane tokeny (balanceOf[A] == 100), adres B może rozdysponować 10 z nich(allowance[A][B] == 10).

Funkcje:

a) transfer - pozwana wysyłać swoje tokeny (z adresu właściciela)
b) approve - informuje, czy osoba B może dysponować określoną ilością tokenów osoby A
c) approveAndCall - działa tak jak approve, tylko przekazuje dodatkowe informacje
d) transferFrom - pozwala wysyłać tokeny z innego adresu (jeśli możemy dysponować odpowiednią ilością tokenów drugiego adresu -> odpowiedni wpis w tablicy allowance)
e) mintToken - pozwala “wykuć” tokeny, wysyłając je potem pod wskazany adres. Funkcji tej może użyć tylko właściciel kontraktu. Można usunąć tą funkcję jeśli w kontrakcie ma być z góry określona ilość tokenów (tak jak w bitcoinie). Wtedy ilością tokenów jest tylko initialSupply z konstruktora
f) setPrices - pozwala ustawić ceny kupna tokenów (w gazie)
g) buy, sell - pozwalają na kupowanie i sprzedawanie tokenów (uwaga, jeśli cena 1 tokena wynosi 100, i zapłacimy 220, to dostaniemy 2 tokeny i 20 zostaje na koncie właściciela kontraktu). Funkcja buy nie pozwoli kupić nam tokenów jeśli wszystkie

zostały już sprzedane (jeśli będzie 1000 tokenów i rozdamy wszystkie to nie można kupić tokenu, do momentu aż ktoś inny nie sprzeda tokenów)

```
contract token is owned {
    /* Public variables of the token */
    string public standard = 'Token 0.1';
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 public totalSupply;

    uint256 public sellPrice;
    uint256 public buyPrice;

    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    /* This generates a public event on the blockchain that will notify clients
*/
    event Transfer(address indexed from, address indexed to, uint256 value);

/* Initializes contract with initial supply tokens to the creator of the
contract */
    function token(
        uint256 initialSupply,
        string tokenName,
        uint8 decimalUnits,
        string tokenSymbol
        ) {
        balanceOf[msg.sender] = initialSupply;
        // Give the creator all initial tokens
        totalSupply = initialSupply;
        // Update total supply
        name = tokenName;
        // Set the name for display purposes
        symbol = tokenSymbol;                                   // Set the symbol
for display purposes
        decimals = decimalUnits;                                // Amount of
decimals for display purposes
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) {
        require (balanceOf[msg.sender] >= _value);           // Check if the
sender has enough
        require (balanceOf[_to] + _value >= balanceOf[_to]); // Check for
overflows
```

```
        balanceOf[_to] += _value;                            // Add the same to
the recipient
        balanceOf[msg.sender] -= _value;                     // Subtract from
the sender
        Transfer(msg.sender, _to, _value);                   // Notify anyone
listening that this transfer took place
    }

    /* Allow another contract to spend some tokens in your behalf */
    function approve(address _spender, uint256 _value)
        returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        return true;
    }

    /* Approve and then communicate the approved contract in a single tx */
    function approveAndCall(address _spender, uint256 _value, bytes _extraData)
        returns (bool success) {
        tokenRecipient spender = tokenRecipient(_spender);
        if (approve(_spender, _value)) {
            spender.receiveApproval(msg.sender, _value, this, _extraData);
            return true;
        }
    }

    /* A contract attempts to get the coins */
    function transferFrom(address _from, address _to, uint256 _value) returns
(bool success) {
        require (balanceOf[msg.sender] >= _value);           // Check if the
sender has enough
        require (balanceOf[_to] + _value >= balanceOf[_to]); // Check for
overflows
        require (_value <= allowance[_from][msg.sender]);    // Check allowance
        balanceOf[_to] += _value;                            // Add the same to
the recipient
        balanceOf[_from] -= _value;                          // Subtract from
the sender
        allowance[_from][msg.sender] -= _value;
        Transfer(_from, _to, _value);
        return true;
    }

    function mintToken(address target, uint256 mintedAmount) onlyOwner {
        balanceOf[target] += mintedAmount;
        totalSupply += mintedAmount;
        Transfer(0, this, mintedAmount);
        Transfer(this, target, mintedAmount);
    }

    function setPrices(uint256 newSellPrice, uint256 newBuyPrice) onlyOwner {
```

```
        sellPrice = newSellPrice;
        buyPrice = newBuyPrice;
    }

    function buy() payable {
        uint amount = msg.value / buyPrice;                // calculates the
amount
        require (balanceOf[this] >= amount);               // checks if it has
enough to sell
        balanceOf[msg.sender] += amount;                   // adds the amount to
buyer's balance
        balanceOf[this] -= amount;                         // subtracts amount
from seller's balance
        Transfer(this, msg.sender, amount);                // execute an event
reflecting the change
    }

    function sell(uint256 amount) {
        require (balanceOf[msg.sender] >= amount );        // checks if the
sender has enough to sell
        balanceOf[this] += amount;                         // adds the amount to
owner's balance
        balanceOf[msg.sender] -= amount;                   // subtracts the
amount from seller's balance
        if (!msg.sender.send(amount * sellPrice)) {        // sends ether to the
seller. It's important
            require(false);                                          // to do
this last to avoid recursion attacks
        } else {
            Transfer(msg.sender, this, amount);            // executes an event
reflecting on the change
        }
    }

    /* This unnamed function is called whenever someone tries to send ether to
it */
    function () {
        assert(false);     // Prevents accidental sending of ether
    }
}
```

KontraktConnections - kontrakt związany z połączeniami. dziedziczy po owned i token (w nawiasie ustawiamy dane do konstruktora kontraktu token)

Zmienne:

a) connectionsCount - ilość połączeń (jako indeks tablicy)

b) soldTicketCount - ilość sprzedanych biletów (jako indeks tablicy)
c) connectionArray - tablica w której przechowywane są dane o konkretnym połączeniu (indeks tablicy jest id połączenia). Tablica zawiera takie informacje: adres ETH twórcy połączenia, ilość miejsc (zmniejsza się przy kupnie biletu) i cena za bilet (w tokenach)
d) soldTicketsArray - tablica zawierająca informacje o kupionych biletach (soldTicketsArray[adres_kupującego][id_połączenia] = ilość_miejsc)

Funkcje:

a) showConnection - wyświetla informacje o połączeniu (szuka po id). Zwraca: adres właściciela, wolne miejsca, cenę i typ
b) lockSlots - pozwala właścicielowi danego połączenia zablokować miejsce (np. jeśli sprzedał bilety w firmie)
c) buyTicket - pozwala na kupno biletu za tokeny
d) createConnection - funkcja pozwalająca tworzyć trasy (podajemy skąd, dokąd, ilość wszystkich miejsc w pojeździe, ilość zajętych miejsc, cenę i typ połączenia)

```
contract Connections is owned, token(1000000, "NAME", 0, "FLO") {
    uint public connectionsCount;
    uint public soldTicketsCount;

    struct Connection {
        address creatorWallet;
        uint8 freeSlots;
        uint16 price;
    }

    mapping (uint => Connection) public connectionArray;
    mapping (address => mapping (uint => uint8)) public soldTicketsArray;

    function Connections(){
        connectionsCount = 0;
        soldTicketsCount = 0;
    }

    function showConnection(uint connectionNumber) returns (address, uint8,
uint16) {
        return (connectionArray[connectionNumber].creatorWallet,
connectionArray[connectionNumber].freeSlots,
connectionArray[connectionNumber].price);
    }
```

```
    function lockSlots(uint connectionNumber, uint8 slotsToLock){
        require (connectionNumber < connectionsCount);
        require (connectionArray[connectionNumber].creatorWallet == msg.sender);
        require (connectionArray[connectionNumber].freeSlots >= slotsToLock);
        connectionArray[connectionNumber].freeSlots =
connectionArray[connectionNumber].freeSlots - slotsToLock;
    }

    function buyTicket(uint connectionNumber, uint8 slots){
        require(freeSlots >= slots);
        require (slots > 0);

        uint tokenAmount = connectionArray[connectionNumber].price*slots;
        address creatorWallet = connectionArray[connectionNumber].creatorWallet;
        uint8 freeSlots = connectionArray[connectionNumber].freeSlots;

        transfer(creatorWallet, tokenAmount);

        soldTicketsArray[msg.sender][connectionNumber] += slots;
        connectionArray[connectionNumber].freeSlots -= slots;
        soldTicketsCount++;
    }

    function createConnection(address carrierWallet, bytes32 from, bytes32 to,
uint8 allSlots, uint8 occupiedSlots, uint16 price){
        require(allSlots >= occupiedSlots);
        bytes32 placeholder;
        placeholder = from;
        placeholder = to;
        placeholder = placeholder;
        connectionArray[connectionsCount].freeSlots = allSlots - occupiedSlots;
        connectionArray[connectionsCount].creatorWallet = carrierWallet;
        connectionArray[connectionsCount].price = price;
        connectionsCount = connectionsCount + 1;
    }
}
```

Kontrakt pozwala na stworzenie naszych tokenów, kupno, sprzedaż i wymianę między sobą tymi tokenami. Pozwala też właścicielowi na zmianę ceny tokena, co pozwoli na dostosowanie się do rynku. Istnieje funkcja, która pozwala wytworzyć dodatkową pulę tokenów, jeśli na rynku będzie ich za mała ilość (funkcję można usunąć, jeśli projekt z założenia ma mieć z góry określoną pulę tokenów, tak jak w bitcoinie). Wytworzyć nowe tokeny może tylko właściciel kontraktu, osoba która udostępniła go do sieci. Wymiana tokenami odbywa się w zdecentralizowanej sieci Ethereum, wszystkie transakcje są publiczne, każdy może zobaczyć dany transfer znając jego hash. Nie ma możliwości

cofnięcia lub przekierowania transakcji po jej wysłaniu (“zaksięgowaniu”), chyba że kontrakt zwróci błąd, wtedy blockchain wraca do stanu sprzed tej transakcji, a waluta jest zwracana do nadawca, co zapobiega sporej liczbie oszustw. Główna część kontraktu związana jest z połączeniami. Kontrakt pozwala na dodawanie połączeń zawierających takie dane jak miejsce odjazdu, miejsce przyjazdu, ilość miejsc (wszystkich oraz zajętych), cenę w naszych tokenach oraz typ połączenia. Połączenie identyfikowane jest po unikalnym numerze, a jego właścicielem jest adres ustawiony podczas tworzenia połączenia. Właściciel połączenia może w późniejszym czasie zablokować miejsca (np. jeśli sprzeda bilety w swojej firmie). Jeśli w połączeniu są jeszcze dostępne miejsca to możliwe jest zakupienie biletu za nasze tokeny. Każdy zakupiony bilet jest identyfikowany przez adres ethereum osoby, która go kupiła, a właściciel połączenia (przewoźnik) od razu może sprawdzić czy dana osoba jest uprawniona do przejazdu.

Komentarze / pytania / uwagi:

Pytanie co się stanie jak ilość połączeń przekroczy uint256 (takiego typu jest id i indeksy w tablicy)? Solidity nie ma typów, które mogą pomieścić większą liczbę

Na ten moment w blockchainie nie ma żadnej informacji co do daty (technologia blockchaina na to pozwala, nie jest to u nas zaimplementowane), tak że teoretycznie można kupić bilet na połączenie z przedwczoraj. Z drugiej strony to przewoźnik sprzedaje bilety i na stronie będzie data. Można dodać metodę, która pozwoli na odblokowanie kupionych miejsc i “recykling” połączenia. (myślę, że na takim recyklingu można zaoszczędzić trochę gazu i indeksów w tablicy) (druga sprawa to walidacja tego, że połączenie zostało wykonane, żeby nie wyczyścić sobie za wcześnie tablicy i sprzedawać drugie tyle biletów na połączenie). Jeśli mielibyśmy dodać czas, to najlepiej chyba unix timestamp (now(); zwraca ten czas))

Jest problem z typem połączenia. String jest tablicą, a Solidity v0.4.13 jeszcze sobie nie radzi z tablicami w strukturach (https://github.com/ethereum/solidity/issues/2549), tak że na ten moment trzeba chyba zrobić osobego mappinga z numerem połączenia i typem (nie wiem czy się opłaca), albo poczekać na update’a

Przykład ICO: BuseoCoin w oparciu o framework Open Zeppelin dla Flotea

1. token do “obliczeń” (mintable coin) korzystając z implementacji Open Zeppelin
2. kontrakt do ICO korzystając z Open Zeppelin
3. kontrakt ICO powinien mieć określony czas startu, zakończenia, cenę i adres portfela

   Uwaga: Oczywiście chodzi o prostą implementację w celach edukacyjnych a nie o wypuszczenie nowego super coina.

```
pragma solidity 0.4.24;
```

```
import "zeppelin-solidity/contracts/token/ERC20/MintableToken.sol";

contract BuseoCoin is MintableToken {
    string public name = "Buseo Coin";
    string public symbol = "BUSC";
    uint8 public decimals = 18;
}
```

```
pragma solidity 0.4.24;

import "./BuseoCoin.sol";
import "zeppelin-solidity/contracts/crowdsale/emission/MintedCrowdsale.sol";
import
"zeppelin-solidity/contracts/crowdsale/validation/TimedCrowdsale.sol";

contract BuseoCrowdsale is MintedCrowdsale, TimedCrowdsale {
    constructor(
        uint _openingTime,
        uint _closingTime,
        uint _rate,
        address _wallet,
        MintableToken _token
    ) public Crowdsale(_rate, _wallet, _token) TimedCrowdsale(_openingTime,
_closingTime) {

    }
}
```

```
pragma solidity ^0.4.24;

contract RouteManager {
    address public owner;

    struct Stop {
        bytes4 id;
        bytes32 name;
        bytes10 latitude;
        bytes10 longitude;
    }

    Stop[] public stops;
```

```
    constructor() public {
        owner = msg.sender;
    }

    modifier isOwner() {
        require(msg.sender == owner, "must be owner");
        _;
    }

    function addStop(
        bytes4 _id,
        bytes32 _name,
        bytes10 _latitude,
        bytes10 _longitude
    ) public isOwner {
        stops.push(Stop(_id, _name, _latitude, _longitude));
    }

    function getStopId(uint _n) public view returns(bytes4) {
        return stops[_n].id;
    }

    function getStopCount() public view returns(uint) {
        return stops.length;
    }
}
```

```
pragma solidity 0.4.24;

import "zeppelin-solidity/contracts/ownership/Ownable.sol";

contract Roles is Ownable {
    address owner;

    struct Manager {
        address manager;
        uint managerSince;
        string name;
    }

    Manager[] managers;
    mapping (address => uint) managerId;

    modifier onlyManager {
```

```
        require(managerId[msg.sender] != 0, "Only for managers");
        _;
    }

    constructor() public {
        owner = msg.sender;
        addManager(0, "");
        addManager(owner, "owner");
    }

    function addManager(address _targetManager, string _managerName) public
onlyOwner {
        uint id = managerId[_targetManager];
        if (id == 0) {
            managerId[_targetManager] = managers.length;
            id = managers.length++;
        }
        managers[id] = Manager({manager: _targetManager, managerSince: now,
name: _managerName});
    }

    function removeManager(address _targetManager) public {
        require(managerId[_targetManager] != 0);
        uint indexToRemove = managerId[_targetManager];
        for(uint i = indexToRemove; i < managers.length-1; i++) {
            managers[i] = managers[i+1];
        }
        delete managers[managers.length-1];
        managers.length--;
    }
}
```

Bilety Open - może być jako rodzaj zniżki kiedy kupujesz bilet w jednym kierunku a na powrotny na tą samą trasę masz “zniżkę”... Więc temat może dotyczyć zniżek dynamicznych. W kontraktach można ustalić, że jeśli ktoś kupił dany kontrakt, na dany przejazd, to może sprawdzić drugi kontrakt.

Powiązania: tak robią przewoźnicy w ramach własnego systemu. Czyli to mogłoby

Dodawanie Trasy narzędzia dla portali / przewoźników

Funkcje:

1. Wyznaczanie trasy przez google - u nas się nie da precyzyjnie bo każdy pkt jest przystankiem więc rysowana trasa pomiędzy przystankami jest równoznaczne ze zdaniem "nie da się wybierać drogi alternatywnej" - wyznacza najkrótszą. Ale... można to ogarnąć

pktami trasy, które NIE są przystankami. Czyli pkt. który jest przystankiem, pkt który jest przystankiem, pkt typu region.

2. Planer na mapie (też rysowaliśmy podobny na tablicy jako V2), zamiast "krokowego" dodawania trasy.

3. Eksporty do GPSów

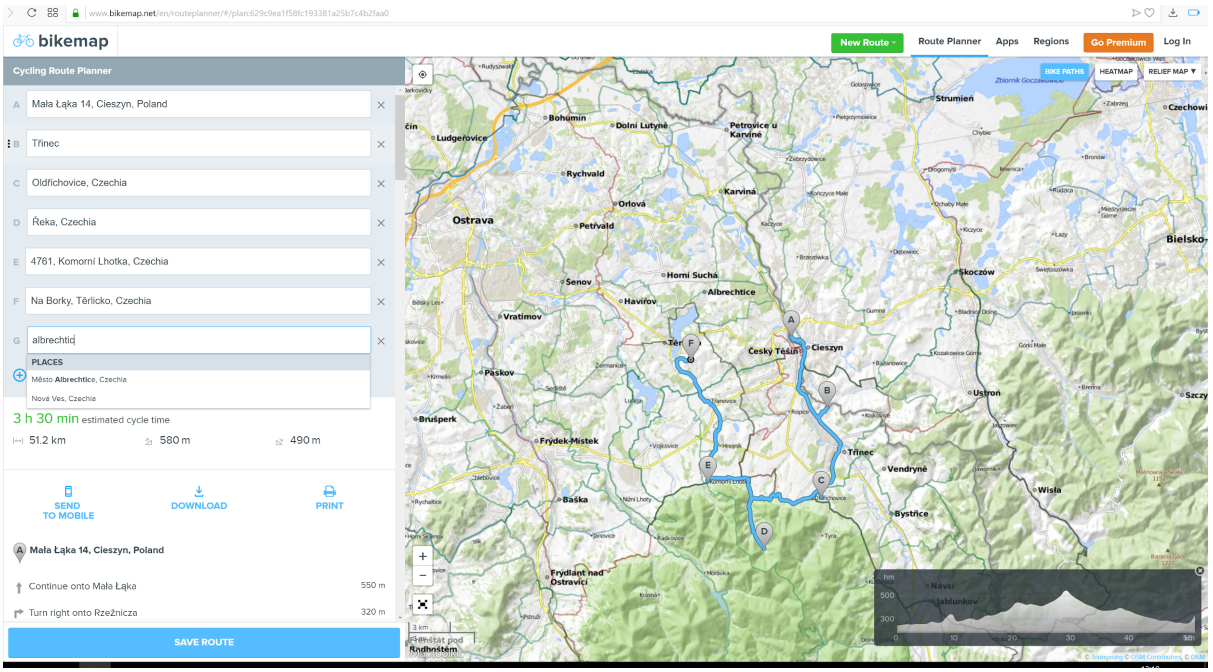
4. Brak Door2Door - zaznaczanie regionów jak do tej pory, ale to i tak nie ma nic wspólnego z kontraktami, bo kontrakt może mieć po prostu "wpis" wprowadzany przez przewoźnika lub pasażera jako opcję tj. "takeMeFrom", "takeMeTo". Gdzie pasasażer wskazuje miejsce odbioru i przywozu (bezużyteczn dla wszystkich typów przejazdów oprocz: 1. taxi 2.uberopodobnych 3. busiarzy doo2door.

5. Rezygnacja z komplikacji niektórych które mamy teraz tj. A. Trasy mają być mało skomplikowane, jeden wpis jedna oferta, jedna trasa. Nie możemy generować tras na pojazdy 9 osobowe, które mają np. 300 odcinków (nie ma sensu)

6. Dodanie trasy powoduje dodanie jej do lokalnej bazy przewoźnika, jest wrzucana do kontraktu.

7. Zarządzanie trasami powinno być zbudowane w formie aplikacji desktop - nie strony WWW (plusy: aplikacja jest na komputerze przewoźnika, nie musi pamiętać strony - szybciej wróci, aplikacja umożliwia szybki dostęp do tras, jako dev - pozbywamy się problemu zależności od przeglądarek typu safari i problemów związanych z wymaganiem typu “musi działać wszędzie tak samo”, Baza danych np. z regionami może znajdować się na komputerze przewoźnika.

Przykład Bike Planner



https://cointelegraph.com/news/amazon-web-services-releases-blockchain-frameworks-for-ethereum-and-hyperledger-fabric
https://neufund.org
https://blog.neufund.org/introducing-typechain-typescript-bindings-for-ethereum-smart-contracts-839fc2becf22
https://docs.google.com/document/d/171b6zvukuuV2UhWLJm9GTrLpIRaW16-xprH-JsRnlfc/edit#
https://daowiki.atlassian.net/wiki/spaces/DAO/pages/2850869/Ether+crypto+currency

Procesy Istniejące w gałęzi Transportu oraz opracowanie standardu

http://fiware-datamodels.readthedocs.io/en/latest/Transportation/doc/introduction/

FloteaRoles.sol

```
pragma solidity 0.4.24;

import "zeppelin-solidity/contracts/ownership/Ownable.sol";

contract FloteaRoles is Ownable {
    address owner;

    struct Manager {
        address manager;
        uint managerSince;
        string name;
    }

    Manager[] managers;
    mapping (address => uint) managerId;

    modifier onlyManager {
        require(managerId[msg.sender] != 0, "Only for managers");
        _;
    }

    constructor() public {
        owner = msg.sender;
        addManager(0, "");
        addManager(owner, "owner");
    }

    function addManager(address _targetManager, string _managerName)
public onlyOwner {
```

```
        uint id = managerId[_targetManager];
        if (id == 0) {
            managerId[_targetManager] = managers.length;
            id = managers.length++;
        }
        managers[id] = Manager({manager: _targetManager, managerSince:
now, name: _managerName});
    }

    function removeManager(address _targetManager) public {
        require(managerId[_targetManager] != 0);
        uint indexToRemove = managerId[_targetManager];
        for(uint i = indexToRemove; i < managers.length-1; i++) {
            managers[i] = managers[i+1];
        }
        delete managers[managers.length-1];
        managers.length--;
    }
}
```

Plik FloteaRoutes.sol

```
pragma solidity ^0.4.24;

import "FloteaRoles.sol";

/// Based on GTFS standard
contract FloteaRoutes is FloteaRoles {
    // FunBus,The Fun
Bus,http://www.thefunbus.org,America/Los_Angeles,(310) 555-0222,en
    struct Agency {
        uint16 agencyId; //~65k
        bytes2 agencyName;
        string agencyUrl;
        bytes32 agencyTimezone;
    }

    //S1,Mission St. & Silver Ave.,The stop is located at the southwest
corner of the intersection.,37.728631,-122.431282,,,
    struct Stop {
        bytes4 stopId; //A1 - ZZ99
        bytes32 stopName;
        bytes10 stopLat; //cheaper than int32
        bytes10 stopLon;
    }
```

```
    //A,17,Mission,"The ""A"," route travels from lower Mission to
Downtown.",3
    struct Route {
        bytes4 routeId;
        uint16 agencyId;
        bytes16 routeShortName;
        bytes32 routeLongName;
        uint8 routeType; //cheaper than bytes2
    }

    //A,WE,AWE1
    struct Trip {
        bytes2 routeId;
        bytes2 serviceId;
        bytes6 tripId; //could be longer: i.e ABWE10
    }

    // AWE1,06:10,06:10,S1,1
    struct StopTime {
        bytes6 tripId;
        bytes5 arrivalTime; // "06:15"
        bytes5 departureTime;
        bytes4 stopId;
        uint8 stopSequence;
    }

    // WE,0,0,0,0,0,1,1,20060701,20060731
    struct Calendar {
        bytes2 serviceId;
        bool monday; //reading cost ~500
        bool tuesday;
        bool wednesday;
        bool thursday;
        bool friday;
        bool saturday;
        bool sunday;
        bytes6 start_date; //YYMMDD
        bytes6 end_date; //YYMMDD
    }

    // WD,20060703,2
    struct CalendarDates {
        bytes2 serviceId;
        bytes6 date;
        uint8 exception_type; // could be bool but uint is cheaper
```

```
    }

    // 1,1000,USD,0,0
    struct FareAttributes {
        uint32 fareId;
        uint24 price; //with decimals so 10 USD is stored as 1000
        bytes4 currency_type;
        uint8 payment_method;
        uint8 transfers;
    }

    // a,TSW,1,1,
    struct FareRules {
        uint32 fareId;
        bytes4 routeId;
    }

    Agency[] public agencies;
    Stop[] public stops;
    Route[] public routes;
    Trip[] public trips;

    function addAgency(
        uint16 _agencyId,
        bytes2 _agencyName,
        string _agencyUrl,
        bytes32 _agencyTimezone
    ) public onlyManager {
        agencies.push(Agency(_agencyId, _agencyName, _agencyUrl,
_agencyTimezone));
    }

    function addStop(
        bytes4 _id,
        bytes32 _name,
        bytes10 _latitude,
        bytes10 _longitude
    ) public onlyManager {
        stops.push(Stop(_id, _name, _latitude, _longitude));
    }

    function addRoute(
        bytes4 _routeId,
        uint16 _agencyId,
        bytes16 _routeShortName,
```

```
        bytes32 _routeLongName,
        uint8 _routeType
    ) public onlyManager {
        routes.push(
            Route(
                _routeId,
                _agencyId,
                _routeShortName,
                _routeLongName,
                _routeType
                )
            );
    }

    function addTrip(
        bytes2 _routeId,
        bytes2 _serviceId,
        bytes6 _tripId
    ) public onlyManager {
        trips.push(Trip(_routeId, _serviceId, _tripId));
    }
}
```

1. Analiza Konkurencji

https://issuu.com/tkidinalog/docs/persbericht_blockchain_9-11-2016
https://docs.google.com/document/d/171b6zvukuuV2UhWLJm9GTrLpIRaW16-xprH-JsRnlfc/edit#
https://skift.com/2018/01/25/travel-megatrends-2018-blockchain-will-spark-a-new-type-of-tech-race-in-travel/
https://www.investopedia.com/news/6-companies-using-blockchain-change-travel-0/

1. https://windingtree.com
2. https://travelchain.io
3. https://toacoin.com
4. https://www.travelcoins.io
5. https://bitair.io
6. https://www.pally.co
7. https://explorecoin.io
8. https://siousada.com
9. https://fujinto.io
10. http://zoomaway.ca

11. https://www.briskpass.com
12. http://lazooz.org
13. https://dovu.io
14. https://shipchain.io
15. https://thecargocoin.com
16. https://mvlchain.io
17. https://www.deepaero.com
18. https://bitnautic.io
19. https://www.300cubits.tech
20. https://boatpilot.io
21. https://www.blockshipping.io
22. https://www.helbizcoin.io
23. https://mcfly.aero/token
24. https://arcade.city
25. https://cartaxi.io

Bilety:

1. https://guts.tickets
2. https://aventus.io
3. https://blocktix.io (bilety na eventy)