Skład zespołu:
Berla Roman
Maciejewski Grzegorz
Pagacz Patrycja
Szewczyk Dawid

1. Elementy systemu

Serwer - technologia Spring Boot

- tu jest algorytm wyboru kierowcy dla danego zamówienia
- tu jest wyliczana cena za przejazd na podstawie obliczonej odległości i taryf (z bazy)

Aplikacja firmowa - WPF

- komunikuje się z Google Maps API
- Funkcjonalność do analitycznego przetwarzania danych zgromadzonych w bazie danych.
    - Przetwarzanie danych pobranych z bazy danych aby poszerzać bazę analityczną (hurtownię)

Aplikacja mobilna na Androida

- komunikuje się z Google Maps API
- komunikuje się z modułem płatności
- komunikacja z modułem autoryzacji

2. Bazy danych

Baza danych obsługi przejazdów

W bazie danych modułu obsługi przejazdów są przechowywane wszystkie informacje:

- Przejazdy **Rides**,
- Klienci **Customers**,
- Metody płatności **Customer_payment_methods**,
- Oceny **Customer_assesments**.
- Kierowcy **Drivers**
- Pojazdy **Vehicles**
- Punkty taryfowe **Ride_points**
- Taryfy **Prices**

Klient chcąc odbyć przejazd tworzy zapytanie **ride_requests**. Składa się ono z pewnej trasy **rider_routes**, w której skład wchodzą pewne punkty **ride_points**.

Posiada ono przypisanego kierowcę **drivers** i pojazd **vehicles**.
Po zakończonym przejeździe zgodnie z metodą płatności odbywa się pobranie opłaty za przejazd payments. System będzie wspierał wiele metod zapłaty payment_methods. Każdorazowa zmiana statusu płatności jest odnotowywana w systemie payment_events.

Baza analityczna

Baza analityczna służyłaby gromadzeniu przetworzonych w odpowiedni sposób danych z bazy funkcjonalnej/produkcyjnej - w której gromadzone są dane podczas codziennego funkcjonowania firmy.
Przykładowe dane która mogłaby zawierać baza analityczna to:

- Sumaryczne przebiegi odpowiednich samochodów
- Dane o średnich przejazdach per auto
- Analityczne dane kierowców
- Analityczne dane pojazdów w korelacji z danymi geograficznymi przejazdów

Baza analityczna byłaby uzupełniana poprzez odpowiedni moduł w aplikacji firmowej, służący do wyświetlania oraz analizowania tych danych.
Z poziomu tego modułu możliwy byłby również eksport tych danych, w celu ich dalszego przetworzenia w programach analitycznych.

Baza analityczna byłaby podłączona bezpośrednio do aplikacji firmowej, a obliczenia niezbędne do jej zasilania będą wykonywane po stronie aplikacji. Aby zasilić bazę analityczną aplikacja wysłałaby żądanie wymaganych do obliczeń danych do serwera, który udostępni dane pochodzące z produkcyjnej bazy danych.

Aktualizacja bazy analitycznej byłaby inicjowana poprzez działanie użytkownika bądź w stałych odstępach czasu. W przypadku stałych odstępów czasu, użytkownik musiałby określić jakie dane mają być aktualizowane.

3. Procesy w systemie

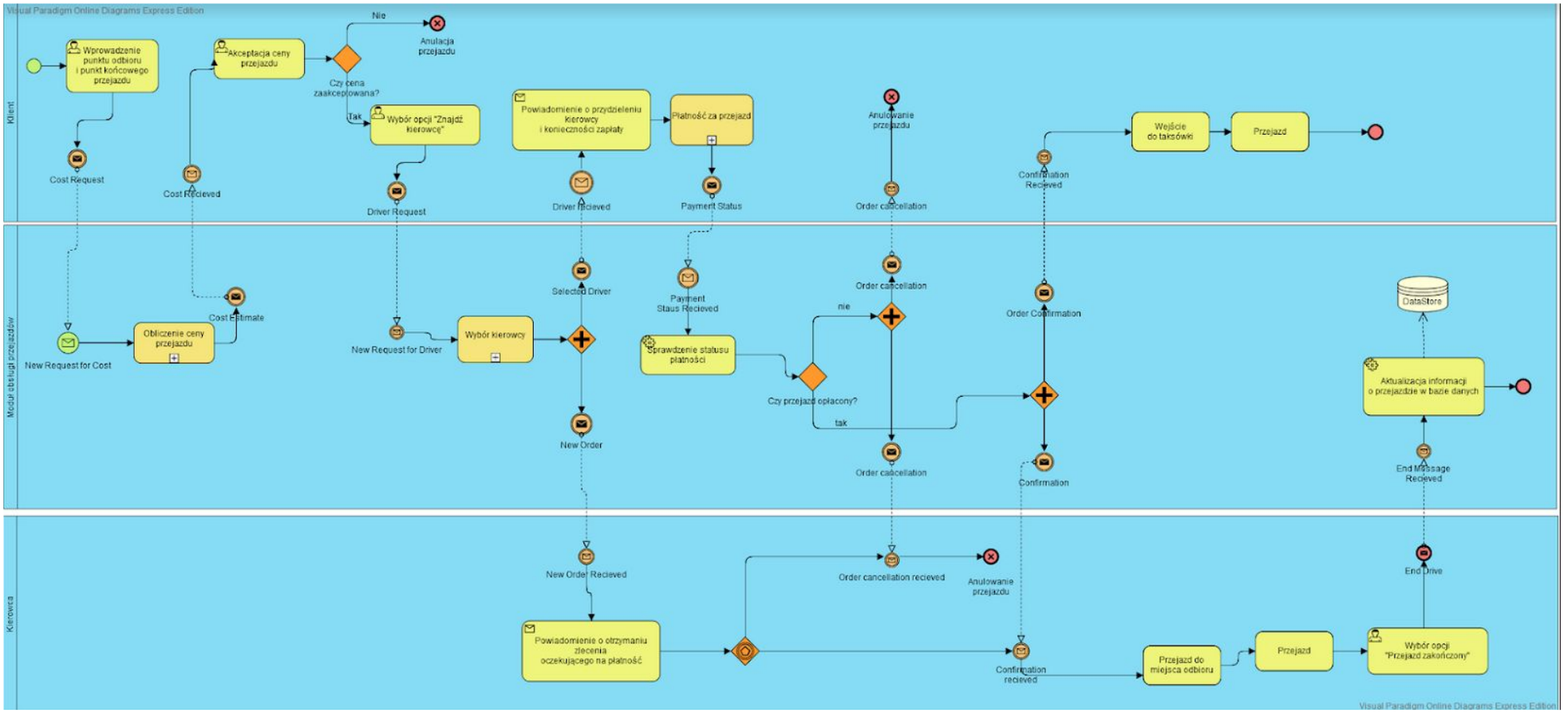
Procesy biznesowe, które możemy modelować:
1. Zamówienie przejazdu przez klienta
2. Wyznaczanie kierowcy który ma odbyć przejazd
3. Rejestracja nowego klienta w systemie
4. Obliczanie ceny przejazdu
5. Płatność za przejazd

#001 Zamówienie przejazdu

Start procesu	Użytkownik wybiera opcję “Zamów przejazd”
Zakończenie procesu	Kierowca wybiera opcję "Zakończ przejazd"
Uczestnicy procesu	1. Klient 2. Moduł obsługi przejazdów (serwer) 3. Kierowca
Przebieg procesu - scenariusz główny	1. Klient wybiera opcję “Zamów przejazd” 2. Klient wprowadza punkt odbioru i punkt końcowy przejazdu 3. Informacje dotyczące planowanej trasy zostają wysłane do serwera. 4. Obliczenie ceny przejazdu. **[Podproces #002]** 5. Wysłanie informacji o przewidywanym koszcie przejazdu do klienta. 6. Akceptacja ceny przejazdu 7. Klient wybiera opcję "Znajdź kierowcę". 8. Nowe zlecenie wyszukiwania kierowcy zostaje wysłane do serwera. 9. Kierowca zostaje przydzielony do przejazdu. **[Podproces #003]** 10. Klient otrzymuje powiadomienie o przydzieleniu kierowcy i konieczności zapłaty. 11. Kierowca otrzymuje informację o nowym zleceniu oczekującym na płatność. 12. Płatność za przejazd.**[Podproces #004]** 13. Wysłanie informacji o statusie płatności do serwera. 14. Sprawdzenie statusu płatności 15. Informacje o przejeździe zostają zapisane w bazie danych. 16. Wysłanie informacji o akceptacji przejazdu do klienta 17. Wysłanie informacji o opłaceniu przejazdu do wybranego kierowcy. 18. Kierowca jedzie we wskazane miejsce odbioru. 19. Klient wsiada do taksówki 20. Odbywa się przejazd. 21. Po dotarciu do celu klient lub kierowca wybiera opcję "Zakończ przejazd". 22. Aktualizacja informacji o przejeździe w bazie danych.
Scenariusz alternatywny i rozszerzenia	6.A Klient nie akceptuje zaproponowanej ceny przejazdu. 7.A Anulowanie przejazdu ---------- 16.B Wysłanie informacji o anulacji przejazdu do klienta 17.B Wysłanie informacji o anulacji przejazdu do kierowcy 18.C Anulacja przejazdu

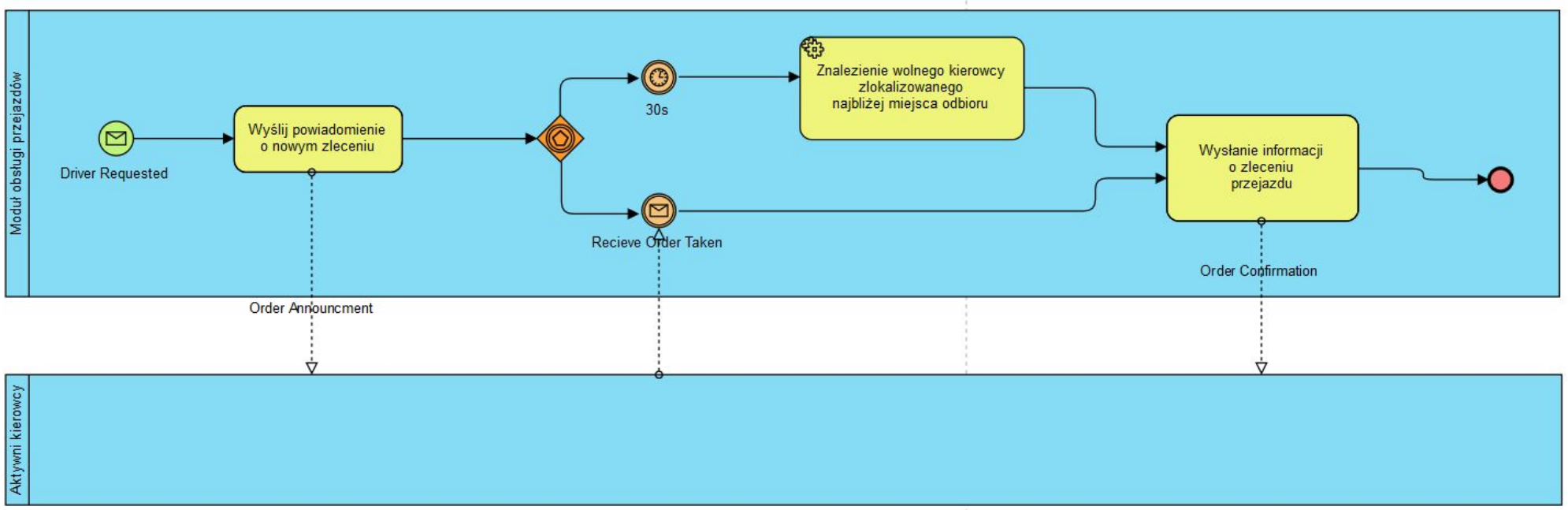
BPMN dla procesu (po poprawkach)

https://drive.google.com/file/d/1c3aL2pglTyvf3yUSstTfKybpLokzqpN9/view?usp=sharing


Visual Paradigm Online Diagrams Express Edition
Klient
Wprowadzenie punktu odbioru i punkt końcowego przejazdu
Akceptacja ceny przejazdu
Nie
Anulacja przejazdu
Czy cena zaakceptowana?
Tak
Wybór opcji "Znajdź kierowcę"
Powiadomienie o przydzieleniu kierowcy i konieczności zapłaty
Płatność za przejazd
Anulowanie przejazdu
Wejście do taksówki
Przejazd
Cost Request
Cost Recieved
Driver Request
Driver Recieved
Payment Status
Order cancellation
Confirmation Recieved
Moduł obsługi przejazdów
Cost Estimate
Obliczenie ceny przejazdu
New Request for Cost
New Request for Driver
Wybór kierowcy
Selected Driver
New Order
Payment Staus Recieved
Sprawdzenie statusu płatności
Czy przejazd opłacony?
nie
tak
Order cancellation
Order Confirmation
Confirmation
DataStore
Aktualizacja informacji o przejeździe w bazie danych
End Message Recieved
Kierowca
New Order Recieved
Powiadomienie o otrzymaniu zlecenia oczekującego na płatność
Order cancellation recieved
Anulowanie przejazdu
Confirmation recieved
Przejazd do miejsca odbioru
Przejazd
Wybór opcji "Przejazd zakończony"
End Drive
Visual Paradigm Online Diagrams Express Edition

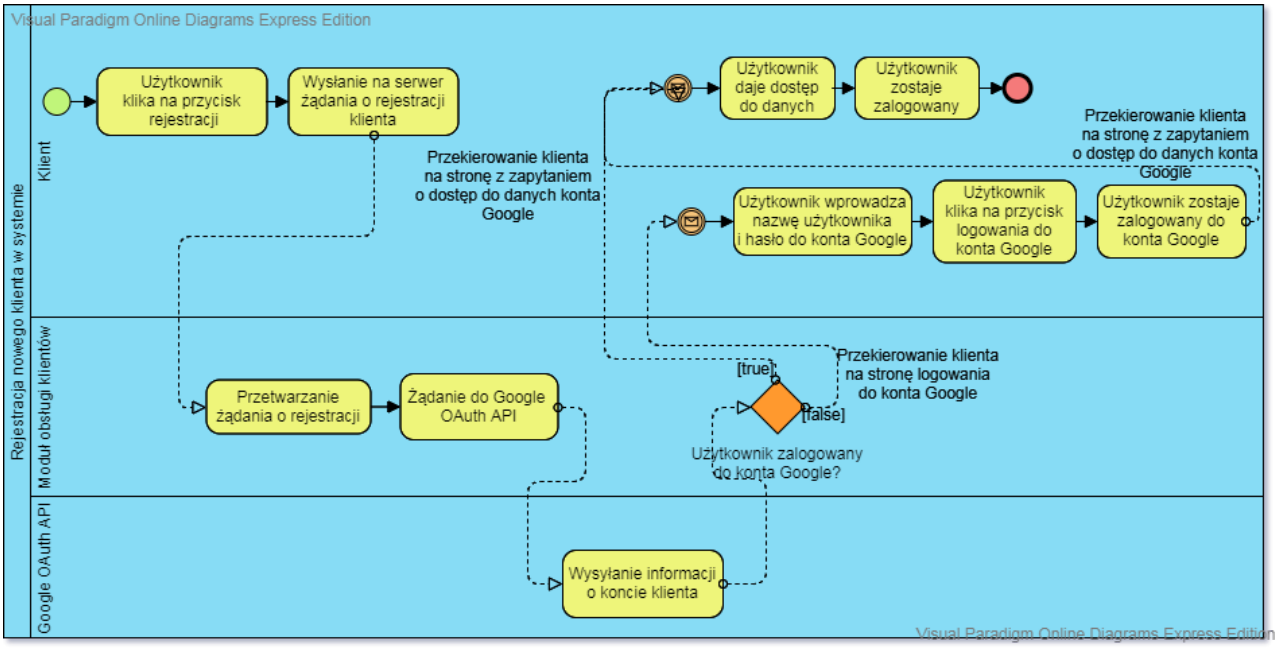
#002 Wybór kierowcy, który ma odbyć przejazd

Start procesu	Serwer otrzymuje informacji o nowym zleceniu przejazdu.
Zakończenie procesu	Zleceniu zostaje przypisany kierowca
Uczestnicy procesu	1. Klient 2. Moduł obsługi przejazdów (serwer) 3. Kierowca
Przebieg procesu	1. Serwer otrzymuje informacji o nowym zleceniu przejazdu. 2. Serwer wysyła powiadomienie o pojawieniu się nowego zlecenia do aplikacji kierowców. 3. Przydział kierowcy a. Kierowca sam podejmuje zlecenie b. Jeśli po upływie 30 s zlecenie nie zostanie podjęte system automatycznie wybiera kierowcę - tego który znajduje się najbliżej miejsca odbioru i jest wolny.


Moduł obsługi przejazdów
Driver Requested
Wyślij powiadomienie
o nowym zleceniu
30s
Znalezienie wolnego kierowcy
zlokalizowanego
najbliżej miejsca odbioru
Recieve Order Taken
Wysłanie informacji
o zleceniu
przejazdu
Order Confirmation
Order Announcment
Aktywni kierowcy

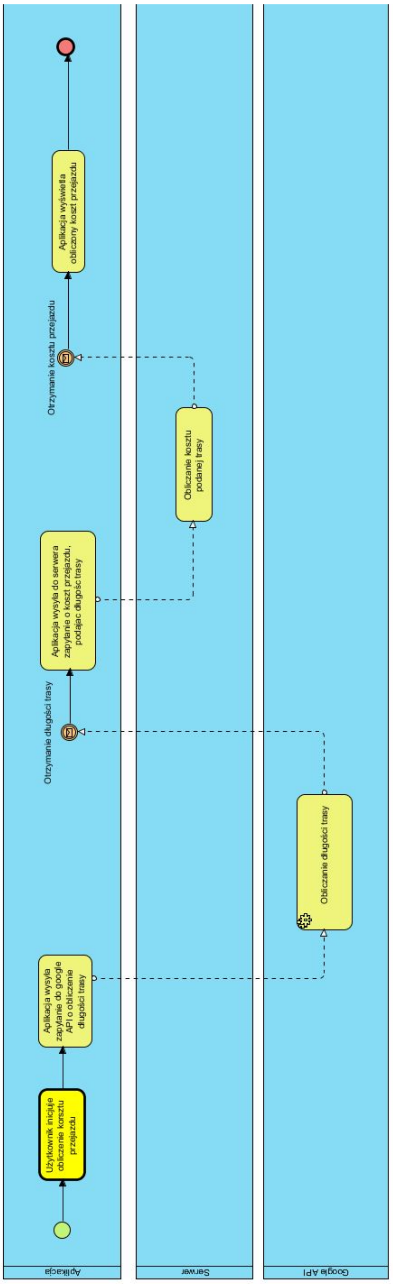
#003 Rejestracja nowego klienta w systemie

Start procesu	Klient chce się zarejestrować
Zakończenie procesu	Klient zostaje zalogowany
Uczestnicy procesu	● Klient (Aplikacja mobilna) ● Moduł obsługi klientów (serwer) ● Google OAuth API
Przebieg procesu - scenariusz główny	1. Klient klika na przycisk rejestracji w aplikacji mobilnej 2. Aplikacja wysyła na serwer żądanie o rejestracji klienta 3. Użytkownik zostaje przekierowany na stronę z zapytaniem o dostęp do danych konta Google 4. Użytkownik klika na przycisk potwierdzający 5. Użytkownik zostaje przekierowany z powrotem do aplikacji mobilnej 6. Użytkownik zostaje zalogowany
Scenariusz alternatywny i rozszerzenia	3A. (Jeżeli użytkownik nie jest zalogowany do konta Google) a. Użytkownik zostaje przekierowany na stronę logowania do konta Google b. Użytkownik wprowadza nazwę użytkownika i hasło do konta Google c. Użytkownik klika na przycisk logowania do konta Google d. Użytkownik zostaje zalogowany do konta Google **e. [Przejście do 3]**



#004 Obliczanie ceny przejazdu

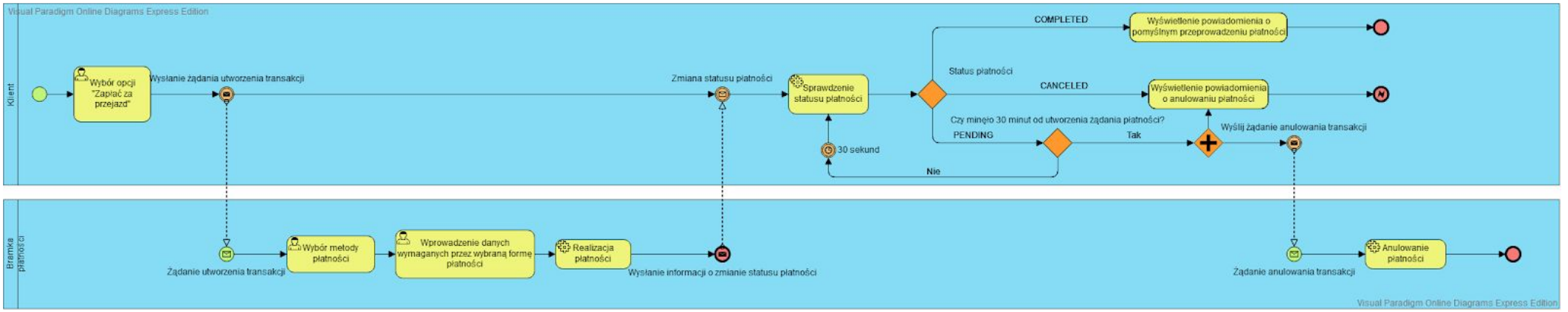
Start procesu	Serwer otrzymuje zlecenie obliczenia kosztu trasy
Zakończenie procesu	Zwrócona zostaje kwota należności
Uczestnicy procesu	● Klient (Aplikacja mobilna) ● Kierowca (Aplikacja firmowa) ● Moduł obsługi przejazdów (serwer) ● Google API
Przebieg procesu	1. Aplikacja otrzymuje żądanie obliczenia kosztu przejazdu 2. Aplikacja wysyła zapytanie do API google w celu obliczenia długości trasy. 3. Google api zwraca długość trasy 4. Aplikacja wysyła do serwera żądanie obliczenia kosztu trasy 5. Serwer otrzymuje żądanie o obliczenie należności za przejazd 6. Serwer oblicza odpowiedni koszt na podstawie długości trasy 7. Serwer zwraca obliczoną należność



#005 Płatność za przejazd

Start procesu	Klient wybiera opcję “Zapłać za przejazd”
Zakończenie procesu	System wysyła klientowi potwierdzenie realizacji płatności.
Uczestnicy procesu	● Klient (Aplikacja mobilna) ● Bramka płatności (PayU API)
Przebieg procesu - scenariusz główny	1. Klient wybiera opcję “Zapłać za przejazd”. 2. System przekierowuje klienta do zewnętrznego systemu płatniczego. 3. Klient wybiera formę płatności i wprowadza wymagane przez nią dane.

	4. Zewnętrzny system płatniczy realizuje płatność. 5. Zewnętrzny system płatniczy wysyła komunikat o pomyślnej realizacji transakcji do aplikacji mobilnej. 6. System wyświetla komunikat o pomyślnej realizacji płatności.
Scenariusz alternatywny i rozszerzenia	5.A. Zewnętrzny system płatniczy wysyła komunikat o anulowaniu transakcji. 6.A. System wyświetla komunikat o niepomyślnej realizacji płatności. ---------- 5.B. Zewnętrzny system płatniczy wysyła komunikat o oczekiwaniu na realizację płatności. 6.B. System wyświetla komunikat o oczekiwaniu na realizację płatności. a. Jeśli zewnętrzny system płatniczy wyśle komunikat o pomyślnej realizacji płatności, następuje przejście do punktu 6 scenariusza głównego. b. Jeśli zewnętrzny system płatniczy wyśle komunikat o niepomyślnej realizacji płatności, następuje przejście do punktu 6.A. scenariusza alternatywnego. c. Jeśli status płatności jest “PENDING” i minęło 30 minut od momentu wysłania żądania realizacji płatności, system wysyła żądanie anulowania płatności do zewnętrznego systemu płatniczego i następuje przejście do punktu 6.A. scenariusz alternatywnego. d. Jeśli status płatności jest “PENDING” i nie minęło 30 minut od momenty wysłania żądania realizacji płatności, system oczekuje 30 sekund, po czym wysyła zapytanie dotyczące statusu płatności do zewnętrznego systemu płatniczego.


Visual Paradigm Online Diagrams Express Edition
Klient
Wybór opcji "Zapłać za przejazd"
Wysłanie żądania utworzenia transakcji
Zmiana statusu płatności
Sprawdzenie statusu płatności
Status płatności
COMPLETED
Wyświetlenie powiadomienia o pomyślnym przeprowadzeniu płatności
CANCELED
Wyświetlenie powiadomienia o anulowaniu płatności
Czy minęło 30 minut od utworzenia żądania płatności?
PENDING
Tak
Nie
30 sekund
Wyślij żądanie anulowania transakcji
Bramka płatności
Żądanie utworzenia transakcji
Wybór metody płatności
Wprowadzenie danych wymaganych przez wybraną formę płatności
Realizacja płatności
Wysłanie informacji o zmianie statusu płatności
Żądanie anulowania transakcji
Anulowanie płatności
Visual Paradigm Online Diagrams Express Edition

Modelowanie procesów integracyjnych w BPMN. Identyfikacja wykorzystywanych struktur wymiany danych, standardów oraz protokołów.

Projekt API

Komunikacja pomiędzy aplikacją mobilną klienta a serwerem HTTP będzie realizowana za pomocą protokołu HTTP (Hypertext Transfer Protocol).

W celu zamodelowania komunikacji między modułami, zebrano funkcje, które te moduły udostępniają sobie wzajemnie.

Pogrupowano je w sześć endpointów, każdy odpowiadający konkretnemu modułowi systemu:

- ride: Moduł obsługi przejazdów
- route: Moduł wyznaczania trasy
- fleet: Moduł zarządzania flotą

API zaprojektowane zostało w programie Swagger, interaktywna dokumentacja znajduje się pod linkami

SWAGGER https://app.swaggerhub.com/apis/isi4/TaxiAPI_2.0/2.0-oas3#trial

BPMN https://drive.google.com/file/d/1aiaw2F4ERcf2IUL-_iel91DKZ7-5FUsM/view?usp=sharing

System płatności - PayU

Komunikacja z systemem PayU odbywa się poprzez wykorzystanie protokołu HTTP, podobnie jak w pozostałych miejscach w systemie. Integracja z bramką płatności opiera się na trzech podprocesach będących częścią procesu nazwanego wyżej "Płatnośc za przejazd". Tymi podprocesami są: tworzenie zamówienia, anulowanie zamówienia i pobranie danych zamówienia (w celu sprawdzenia statusu zamówienia).

Tworzenie zamówienia

Utworzenie zamówienia odbywa się poprzez wykonanie metody POST na endpoint */api/v2_1/orders* systemu PayU.

Komunikat *OrderCreateRequest* musi składać się z pól wymienionych poniżej. Spośród wszystkich pól obsługiwanych przez system PayU wybrane zostały tylko te, które są wymagane przez bramkę płatności lub są istotne z punktu widzenia projektowanego systemu:

- extOrderId - identyfikator płatności systemu TaxiCorp,
- notifyUrl - adres wykorzystywany przez system PayU w celu wysłania powiadomienia (np. dotyczącego zmiany statusu płatności),
- customerIp - adres IP płacącego,
- merchantPosId - identyfikator punktu płatności,
- description - opis zamówienia,
- currencyCode - waluta zamówienia w standardzie ISO 4217,
- totalAmount - całkowity koszt zamówienia w najmniejszej jednostce waluty (np. grosze),

- products - tablica obiektów *product*, każdy z nich musi zawierać pola:
    - name - nazwa produktu,
    - unitPrice - cena jednostkowa,
    - quantity - liczba sztuk.

Powiadomienia z systemu PayU są wysyłane w formacie JSON poprzez wykonanie metody POST na adres podany podczas tworzenia zamówienia w polu *notifyUrl*. Powiadomienia zawierają obiekt typu *order*, który składa się z pól podanych w komunikacie *OrderCreateRequest* oraz dodatkowych:

- orderId - identyfikator płatności nadany przez system PayU,
- orderCreateDate - data utworzenia zamówienia,
- validityTime - czas w trakcie którego możliwe jest dokończenie zamówienia,
- additionalDescription - dodatkowy opis zamówienia (niewykorzystywane przez system TaxiCorp),
- status - status zamówienia,
- buyer - sekcja zawierająca dane kupującego (niewykorzystywana przez system TaxiCorp).

Anulowanie zamówienia

Anulowanie zamówienia odbywa się poprzez wykonanie metody DELETE na endpoint */api/v2_1/orders/{orderId}*. Anulowane może być tylko zamówienie ze statusem innym niż COMPLETED. Odpowiedź systemu PayU zawiera następujące pola:

- orderId - identyfikator zamówienia systemu PayU,
- extOrderId - zewnętrzny identyfikator zamówienia systemu TaxiCorp,
- status - obiekt typu status, zawierający pola:
    - statusCode - kod odpowiedzi,
    - statusDesc - opis statusu odpowiedzi.

Pobranie danych zamówienia

Pobranie zamówienia odbywa się poprzez wykonanie metody GET na endpoint */api/v2_1/orders/{orderId}*. Odpowiedź zawiera obiekt typu order opisany powyżej.

Źródło: http://developers.payu.com/pl/restapi.html

BPMN: https://drive.google.com/file/d/1fcCDAe07buq7Ty8DkbhfBZ2fSfEQfHlB/view?usp=sharing

Obliczanie kosztu przejazdu - Google API

Tak jak zostało to opisane powyżej, częścią procesu obliczania kosztu przejazdu jest wyznaczenie długości przewidywanej trasy.

W procesie składania zamówienia przejazdu klient podaje dane wejściowe jakimi są adres początkowy i docelowy. Dane te są przetwarzane na współrzędne, a same współrzędne trafiają jako argumenty do procesu obliczania należności.

Proces obliczania płatności, w pierwszej kolejności zostanie wysłane zapytanie do google API distance matrix, w celu obliczenia długości trasy:

- https://maps.googleapis.com/maps/api/distancematrix/outputFormat?parameters

Wymaganymi parametrami są

- format wyjściowy
- dane wejściowe
    - origins - punkt(y) początkowy
    - destinations - punkt(y) końcowy
    - Key - klucz aplikacji

Na tak skonstrułowane zapytanie API zwraca plik w określonym formacie (json bądź XML), który zawiera informacje o dystansie i przewidywanym czasie podróży (domyślnie wyliczane dla samochodów).

Możliwe jest także podanie dodatkowych parametrów takich jak ograniczenia względem wyznaczonej trasy (unikanie opłat, autostrad, terenów zabudowanych..).

Z otrzymanego pliku wartość przypisaną do pola “distance” przekazujemy do naszego serwera

- route/amount

Serwer bazując na podanej długości trasy, zwraca należność.

BPMN: https://drive.google.com/open?id=1zIFgcVRpqWUEhoI00VbaepB4HK2mY-nh

Raport zajęcia nr 6 - 06.04.2020

Kod źródłowy projektu:
https://github.com/berl-a/ISIApp/tree/RELEASE_05.04

Poprawiony link do dokumentacji API w Swager
https://app.swaggerhub.com/apis-docs/patipag/TaxiAPI/1.0.0-oas3#

Odpowiedzi na pytania z poprzedniego tygodnia

Kolejkowanie komunikatów - Rabbit MQ

- Klient kupuje towar i oczekuje na zapłatę z zewnętrznego procesora płatniczego. Po uzyskaniu wiadomości z procesora o prawidłowo dokonanej płatności można rozpocząć wysyłkę danych do aplikacji mobilnej, celem zakończenia procesu zamawiania taksówki.
- Serwer (Spring Boot) będzie nadawcą wiadomości, a aplikacja mobilna będzie subskrybowała się na konkretną kolejkę.

XSL - transformata XSLT

Planowane jest wykorzystanie możliwości transformaty XSLT w celu dostosowania otrzymanych plików XML do potrzeb odpowiednich aplikacji.
API Distance Matrix od google dostarcza pliki zarówno XML jak i JSON, transformata XSLT daje możliwość przekształcenia pliku JSON do XML, oraz dostosowanie formatu pliku XML do naszych potrzeb, które są minimalne w stosunku do informacji dostarczanych przez API.

W celu określenia czy dokument XML ma oczekiwany schemat, jest on poddawany walidacji z wykorzystaniem XML Schema. Proces ten odbywa się przed poddaniem dokumentu transformacji XSLT (i jest od niego niezależny), aby zapewnić zgodność pliku wyjściowego ze schematem wykorzystywanym w aplikacji.

Prace implementacyjne

Klient - Aplikacja kierowcy (WPF)
https://bitbucket.org/dawid_szewczyk/isi_taxidriverapp/branches/?status=all

Została stworzona podstawowa struktura aplikacji desktopowej WPF.

Kod źródłowy do tej części jest dostępny na branchu release/Release_01_06042020

Klient - Aplikacja klienta
(Android)https://bitbucket.org/ppagacz/taxiapp/branches/?status=all

Została stworzona podstawowa struktura aplikacji mobilnej android. Dodano aktywność z widokiem mapy Google. Aby możliwe było wykorzystanie Maps SDK for Android wygenerowano klucz autoryzacyjny do Google Api. W pliku `gradle.properties` dodano tak wygenerowany klucz. Mapa poprawnie wyświetla się po uruchomieniu aplikacji na emulatorze.

Kod źródłowy do tej części jest dostępny na branchu release/Release_01_06042020

Server Spring

Serwer aplikacji został napisany w języku Java z użyciem Spring Boot.

Źródłowy kod aplikacji może być znaleziony pod poniższym linkiem:
https://github.com/berl-a/ISIAppBackend

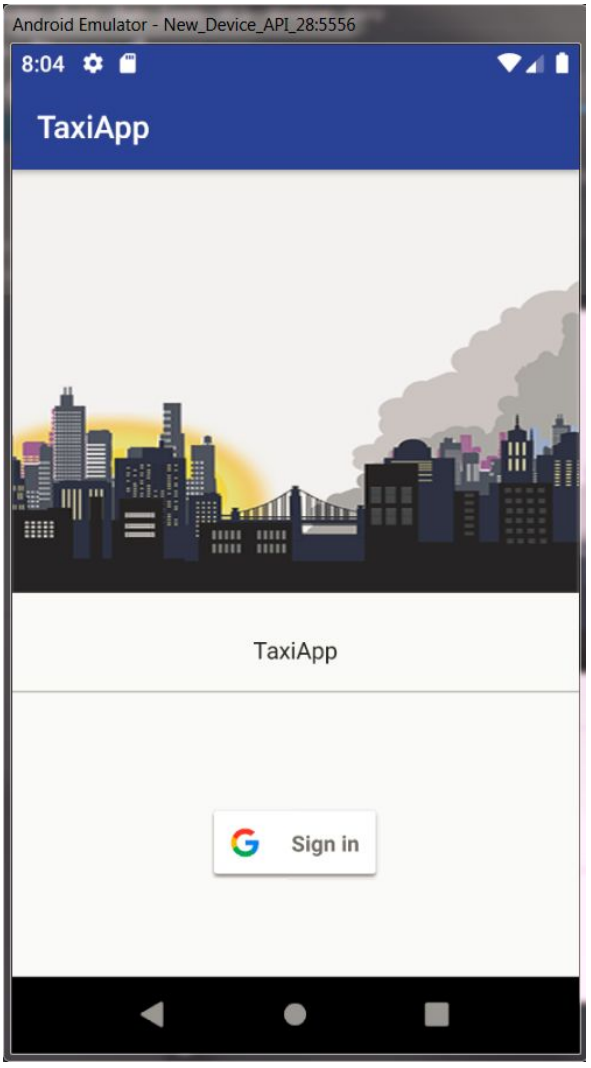
Changelog:

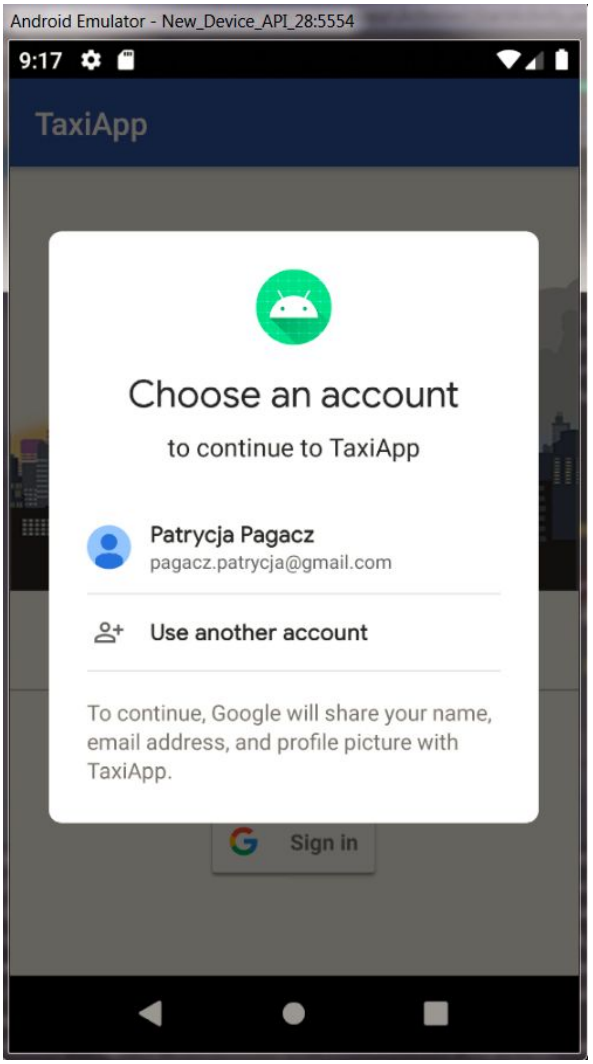
- Add project structure
- Add example controllers, service and DAO classes

Postęp prac implementacyjnych - 20.04.2020

1. **Android**

Dodanie aktywności startowej pokazywanej po uruchomieniu aplikacji. Dodanie przycisku rejestracji za pomocą konta google. Podczas pierwszego logowania się do aplikacji TaxiApp użytkownik musi wybrać konto, którego chce użyć do logowania. Gdy użytkownik loguje się do aplikacji kolejny raz i jest już zalogowany na urządzeniu mobilnym logowanie następuje w sposób automatyczny i ten ekran w ogóle nie jest prezentowany.





Po zalogowaniu się użytkownika do konta google następuje pobranie Id tokena jego konta. W dalszej części ten token będzie wysyłany do endpointa na serwerze, gdzie będzie odbywała się walidacja. Komunikacja z serwerem nie została jeszcze obsłużona w tej części prac.

2. **WPF**

Dodana została część funkcjonalności obliczania ceny przejazdu. Obejmuje ona pobieranie danych z Google Distance Matrix API. Komunikacja z serwerem nie została jeszcze obsłużona na tym etapie prac, więc testy funkcjonalności odbywały się na statycznych danych. Wynik otrzymany

z Google Distance Matrix API jest zapisywany w formacie XML w celu późniejszego wykorzystania w transformacji XSLT.

Utworzone zostały także pliki do obsługi XML schema oraz transformaty XSLT. Transformata XSLT została przetestowana i wpleciona w logikę aplikacji, przetwarzając pliki z wykorzystaniem schematu oraz samej transformaty, uzyskując tym samym na wyjściu plik zawierający wyłącznie informacje które nas interesują.

Server Spring

Serwer aplikacji został napisany w języku Java z użyciem Spring Boot.

Źródłowy kod aplikacji może być znaleziony pod poniższym linkiem:
https://github.com/berl-a/ISIAppBackend

Serwer aplikacji został umieszczony pod adresem:
http://isiapp-env.eba-nhbvhbzn.us-east-1.elasticbeanstalk.com/

Changelog:

- Deploy server
- Add JWT-based security (https://youtu.be/-IUyB6lCDCE instrukcja z obsługi serwera)
- Add getCost method
- Add many CRUD methods