## **Skład zespołu:** Berla Roman Maciejewski Grzegorz Pagacz Patrycja Szewczyk Dawid

## 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

## 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.

## Procesy w systemie

**Procesy biznesowe, które możemy modelować:**

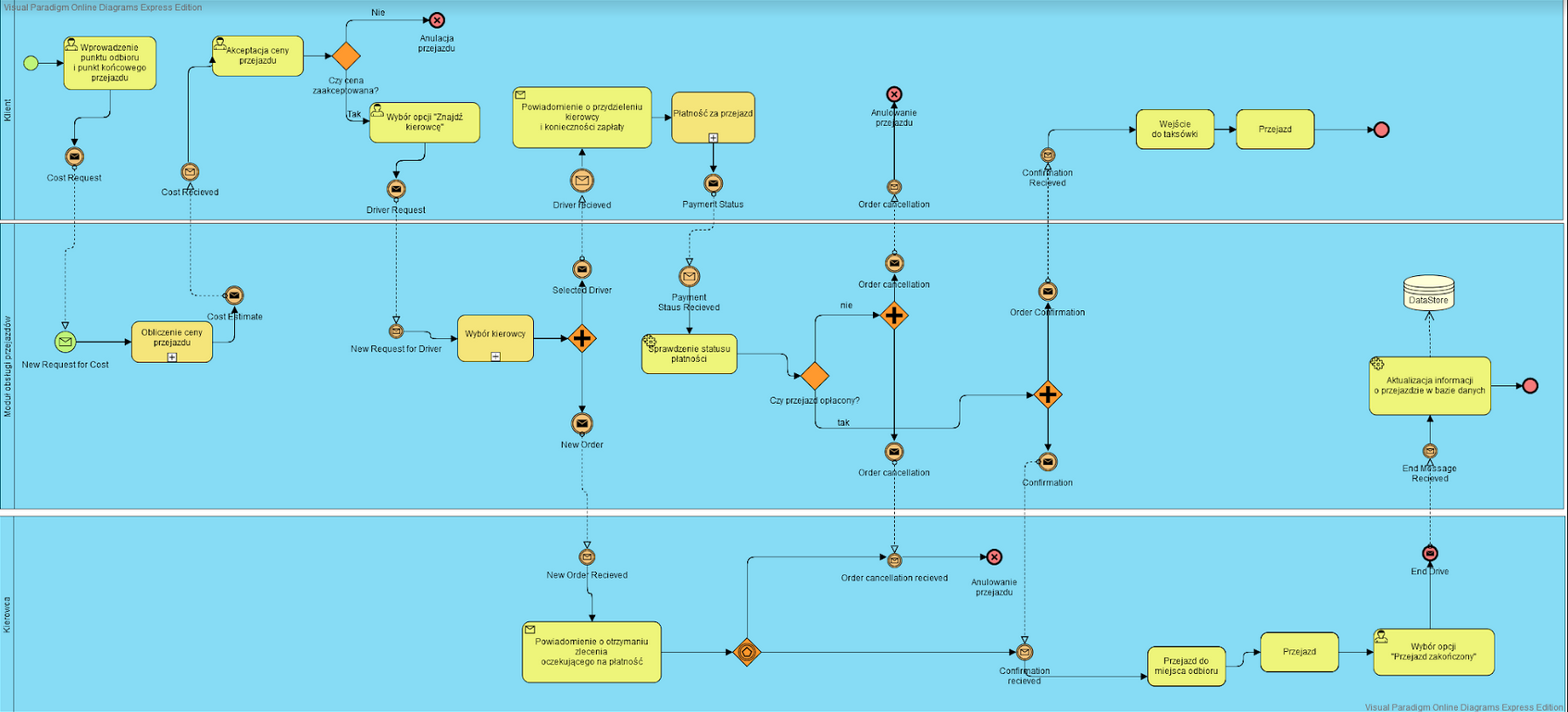
1. Zamówienie przejazdu przez klienta
2. Wyznaczanie kierowcy który ma odbyć przejazd
3. Rejestracja nowego klienta w systemie
4. Obliczanie ceny przejazdu
5. Płatność za przejazd

**#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]**   1. Wysłanie informacji o przewidywanym koszcie przejazdu do klienta. 2. Akceptacja ceny przejazdu 3. Klient wybiera opcję "Znajdź kierowcę". 4. Nowe zlecenie wyszukiwania kierowcy zostaje wysłane do serwera. 5. Kierowca zostaje przydzielony do przejazdu.  **[Podproces #003]** 6. Klient otrzymuje powiadomienie o przydzieleniu kierowcy i konieczności zapłaty. 7. Kierowca otrzymuje informację o nowym zleceniu oczekującym na płatność. 8. Płatność za przejazd.**[Podproces #004]** 9. Wysłanie informacji o statusie płatności do serwera. 10. Sprawdzenie statusu płatności 11. Informacje o przejeździe zostają zapisane w bazie danych. 12. Wysłanie informacji o akceptacji przejazdu do klienta 13. Wysłanie informacji o opłaceniu przejazdu do wybranego kierowcy. 14. Kierowca jedzie we wskazane miejsce odbioru. 15. Klient wsiada do taksówki 16. Odbywa się przejazd. 17. Po dotarciu do celu klient lub kierowca wybiera opcję "Zakończ przejazd". 18. 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 |

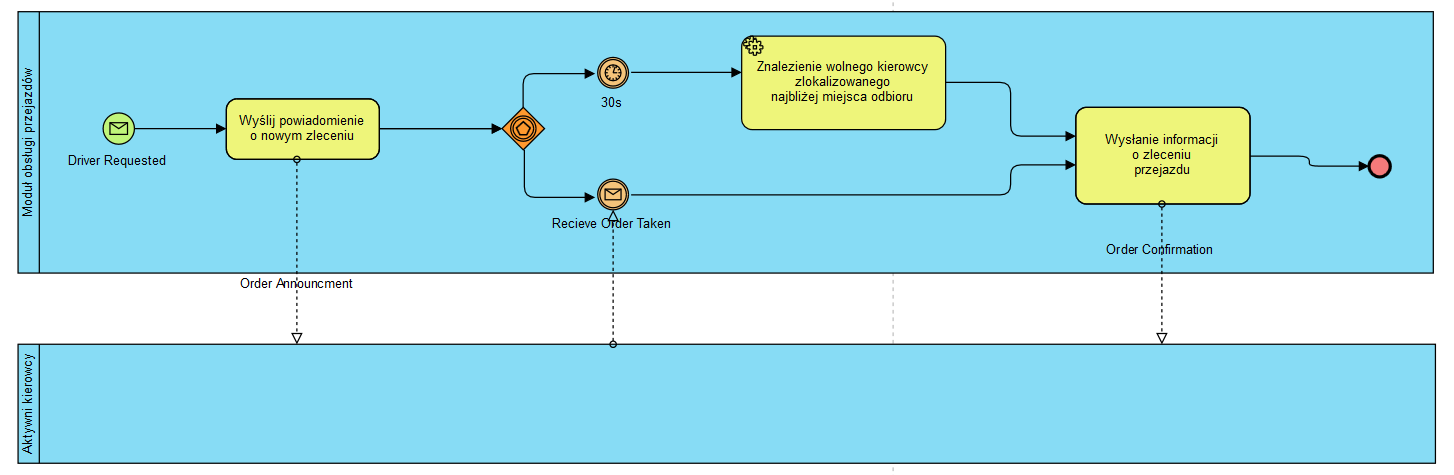
**BPMN dla procesu (po poprawkach)**

[**https://drive.google.com/file/d/1c3aL2pglTyvf3yUSstTfKybpLokzqpN9/view?usp=sharing**](https://drive.google.com/file/d/1c3aL2pglTyvf3yUSstTfKybpLokzqpN9/view?usp=sharing)

****

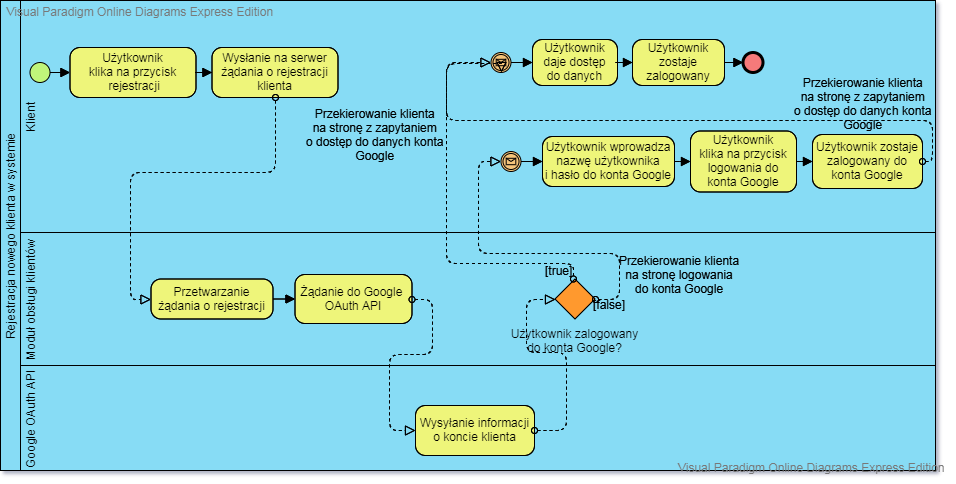
**#002 Wybór kierowcy, który ma odbyć przejazd**

|  |  |
| --- | --- |
| **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    1. Kierowca sam podejmuje zlecenie    2. Jeśli po upływie 30 s zlecenie nie zostanie podjęte system automatycznie wybiera kierowcę -   tego który znajduje się najbliżej miejsca odbioru i jest wolny. |

****

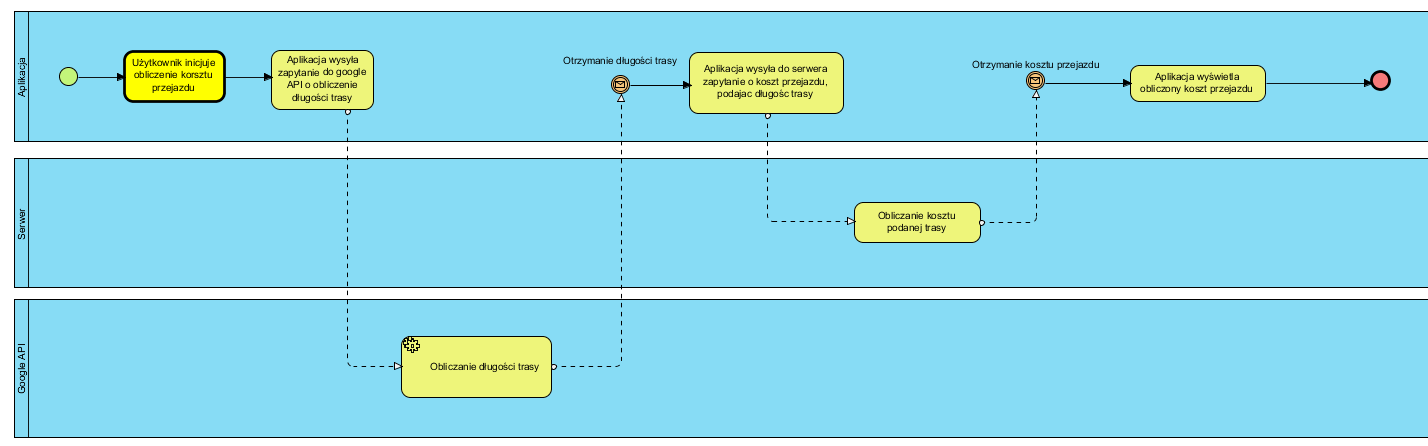
**#003 Rejestracja nowego klienta w systemie**

|  |  |
| --- | --- |
| **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)   1. Użytkownik zostaje przekierowany na stronę logowania do konta Google 2. Użytkownik wprowadza nazwę użytkownika i hasło do konta Google 3. Użytkownik klika na przycisk logowania do konta Google 4. Użytkownik zostaje zalogowany do konta Google 5. **[Przejście do 3]** |

****

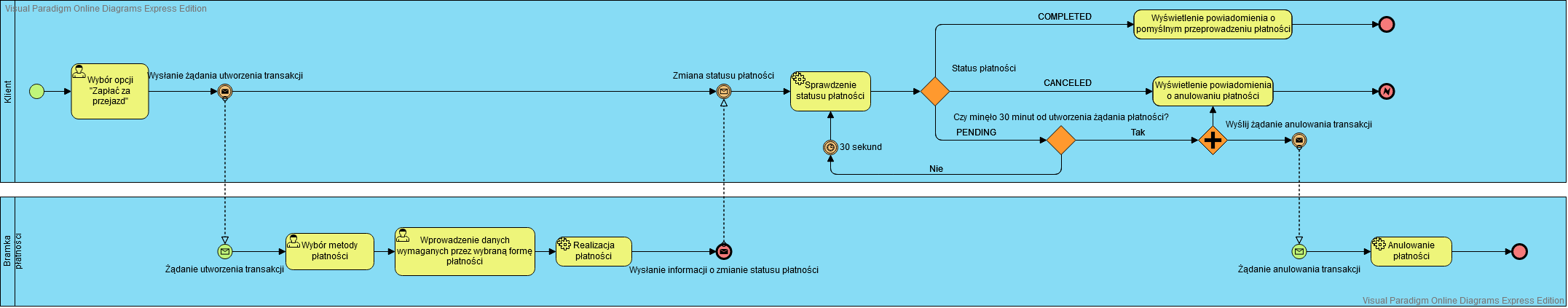
**#004 Obliczanie ceny przejazdu**

|  |  |
| --- | --- |
| **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.   1. Jeśli zewnętrzny system płatniczy wyśle komunikat o pomyślnej realizacji płatności, następuje przejście do punktu 6 scenariusza głównego. 2. Jeśli zewnętrzny system płatniczy wyśle komunikat o niepomyślnej realizacji płatności, następuje przejście do punktu 6.A. scenariusza alternatywnego. 3. 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. 4. 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. |



**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#**](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**](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**](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>

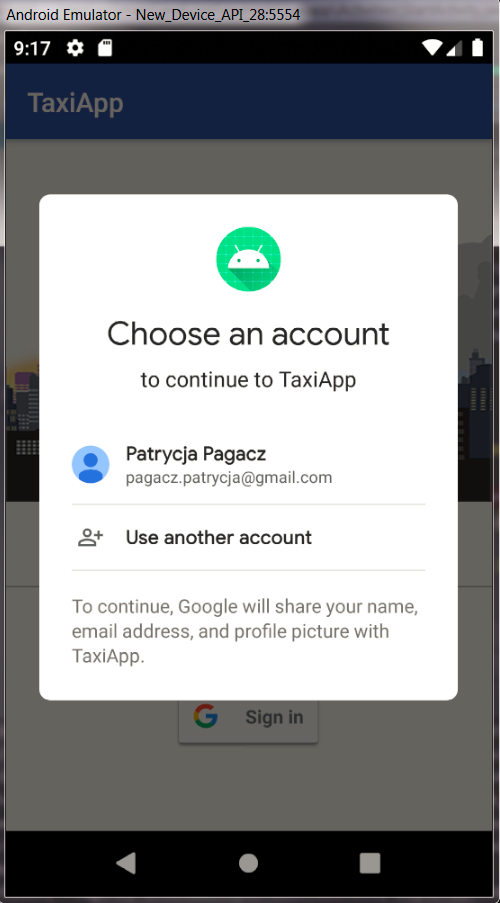
Changelog:

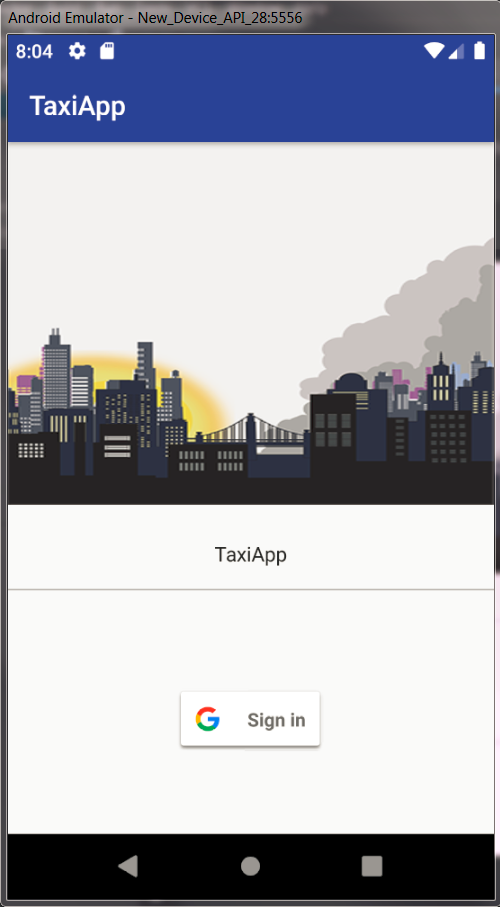
* Add project structure
* Add example controllers, service and DAO classes

### 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.

1. **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

### Postęp prac implementacyjnych - 27.04.2020

1. **WPF**

Została dodana komunikacja z serwerem obejmująca logowanie i pobieranie tokena, który jest wykorzystywany w celu autoryzacji przyszłych żądań. Jest to rozwiązanie tymczasowe i w przyszłości zostanie zmodyfikowane, ponieważ docelowo logowanie ma odbywać się poprzez konto Google.

Została dodana także osobna weryfikacja wejściowego pliku XML przez XML schema, poprzedzająca uruchomienie transformaty XSLT, bądź nie, w zależności od wyniku walidacji dokumentu. Została dopisana także logika wyodrębniająca samą wartość dystansu, zwróconego przez API Distance Matrix, który jest zwracany i gotowy do przesłania na serwer w celu obliczenia należności.

1. **Spring Server**

* Testowanie integracji z frontendem
* Zmiana silnika bazy danych na MySQL
* Postawienie bazy danych na platformie AWS RDS

1. **Android**

Została zrobiona główna aktywność z mapą i menu. Problem, który na razie występuje, to że położenie po otwarciu mapy nie ustawia się z automatu na obecna lokalizację. Problem jest z access permission. W tym tygodniu nie udało nam się go rozwiązać, pracujemy nad nim dalej.

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

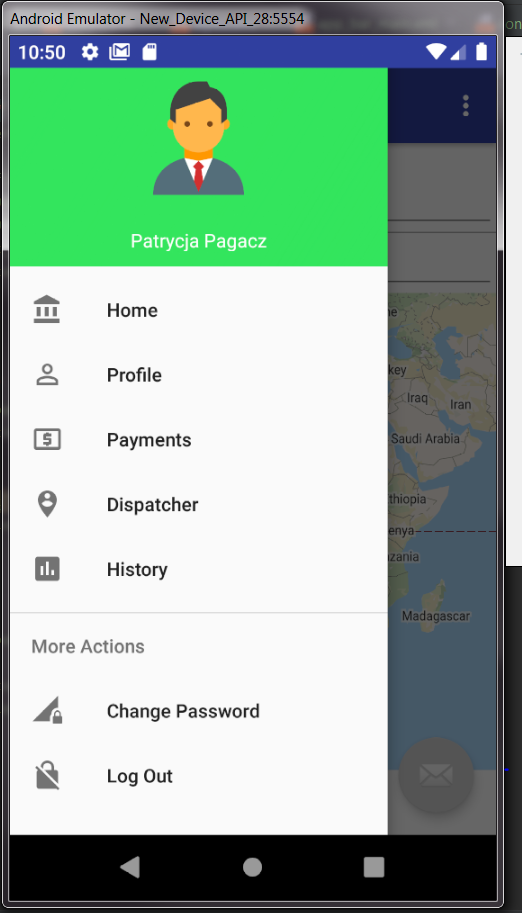
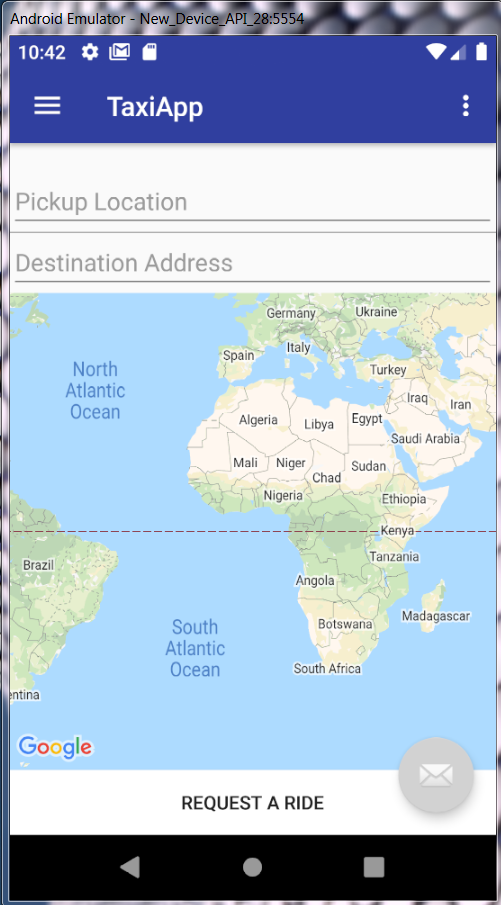
<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

Odpowiedź na pytanie z zeszłego tygodnia.

**1. W części 1 dotyczącej aplikacji na androida - czy po zalogowaniu na pewne konto google istnieje możliwość jego późniejszej zmiany?**

Konto w naszym systemie jest przypisane do jednego ID tokena konta gmail. Jeśli więc na jednym urządzeniu logujemy się dwa razy różnymi kontami gmail powstaną 2 konta w systemie. W aplikacji jest opcja Log Out, która umożliwia wylogowanie i ponowne wybranie konta gmail. Jeśli wybierzemy inne tworzy się nowe konto. Czy w obrębie jednego konta można zmienić konto google, czyli podpiąć inne - takiej funkcjonalności nie przewidujemy.

****

**2.Co oznaczają zdania “Dodana zostałą część funkcjonalności obliczania ceny przejazdu” i “komunikacja z serwerem nie została jeszcze obsłużona”**

Obecnie dodana została tylko część obliczania ceny przejazdu, która ma odbywać w kilku etapach, aplikacja kierowcy wysyła zapytanie do API Distance Matrix otrzymując w odpowiedzi dystans w kilometrach, co jest już zaimplementowane. Następnie dystans jest wysyłany w zapytaniu na serwer w celu obliczenia należności (ride/getCost).

### Postęp prac implementacyjnych - 04.05.2020

1. **WPF**

Zaimplementowana została część logiki odpowiadająca za pobranie dostępnych przejazdów do aplikacji, wyświetlając je w postaci listy, gotowe do podjęcia przez kierowcę. Testy były przeprowadzane na danych lokalnych, docelowe będą one pobierane z serwera - funkcjonalność obejmuje również zapytanie wysyłane do DistanceMatrix api w celu obliczenia dystansu.

Zaimplementowano także pobieranie ceny przejazdu obliczanej na serwerze na podstawie długości przejazdu oraz poprawiono kilka błędów związanych z asynchronicznym przesyłaniem zapytań do serwera.

1. **Aplikacja android**

Udało się zaimplementować to czego nie było w zeszłym tygodniu czyli automatyczne ustawianie położenia na mapie na obecna lokalizacje użytkownika. Wykorzystano w tym celu Places API. Ponadto zrealizowano pierwszy krok procesu “Zamówienie przejazdu” (BPMN dla procesu znajduje się powyżej). Użytkownik może wprowadzić punkt odbioru i końcowy i po naciśnięciu przycisku “Request a ride” zostanie wysłane żądanie na serwer typu POST z następującym body

{

"startLatitude": 0,

"startLongitude": 0,

"destinationLatitude": 0,

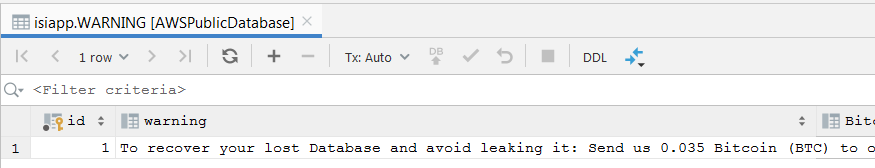
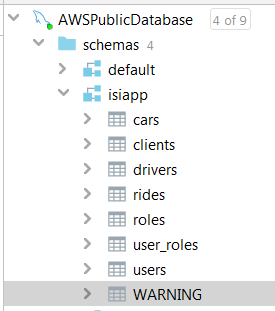
"destinationLongitude": 0

}

Komunikacja z serwerem nie została jeszcze przetestowana.

1. **Backend**

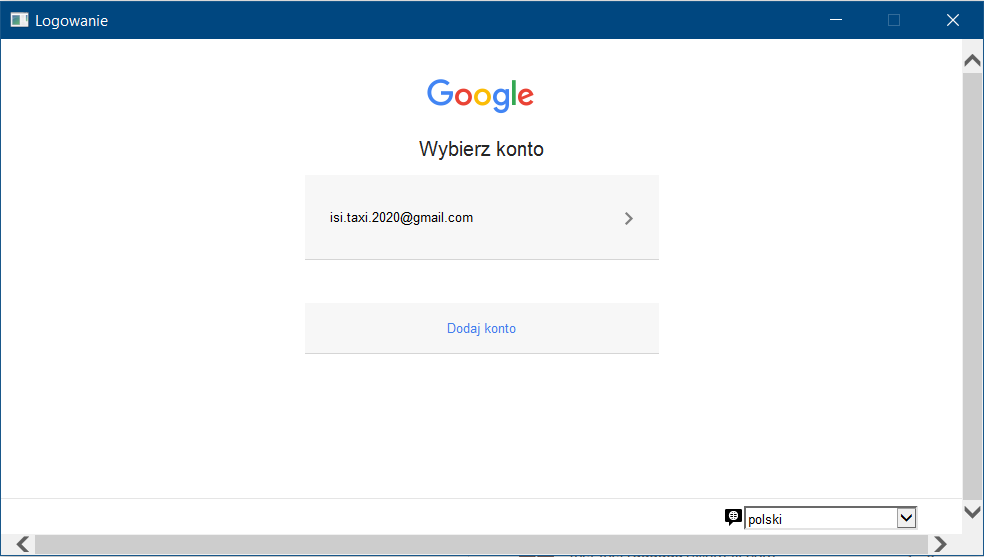
Changelog:

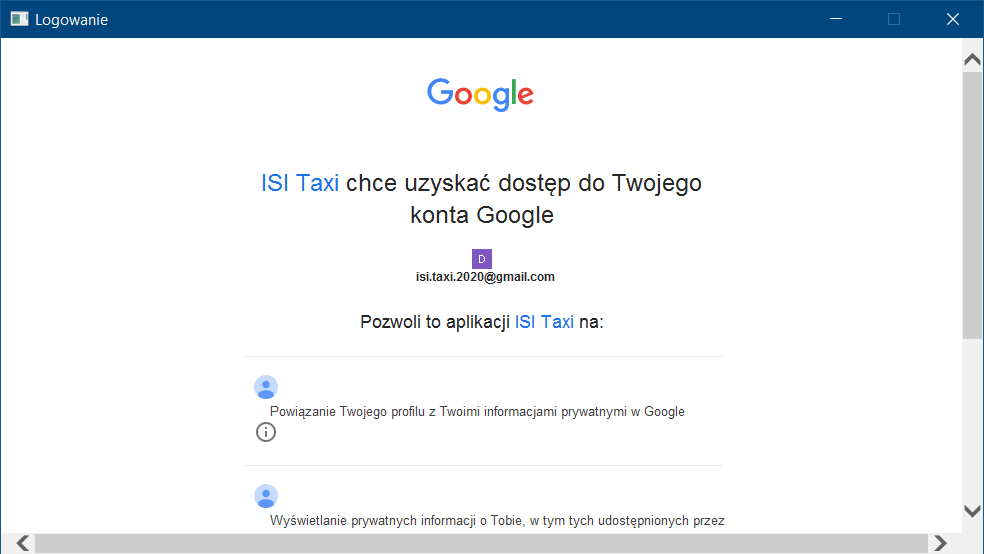
* Zmiana endpointa dla obliczania ceny przejazdu, teraz przyjmuje tylko długość trasy
* Dodanie endpointa dla historii przejazdów
* Usunięcie niepotrzebnych pól z klasy Client
* Recovery po schakowaniu bazy danych  
  
* Wygenerowanie kont w systemie ubezpieczeniowym dla aplikacji Android i WPF

### Postęp prac implementacyjnych - 11.05.2020

1. **WPF**

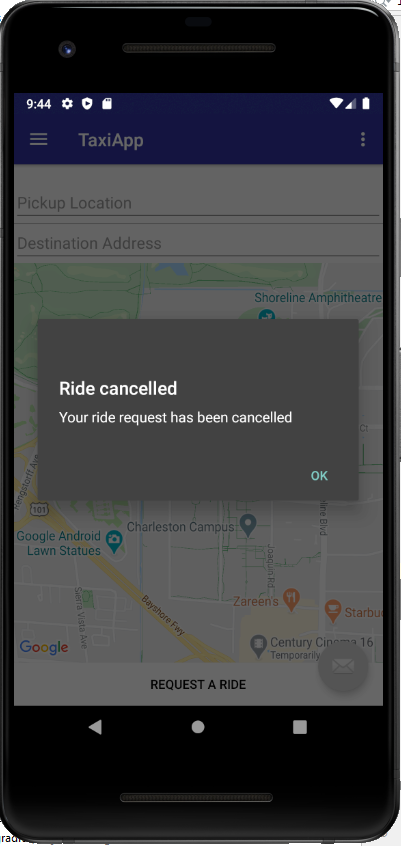
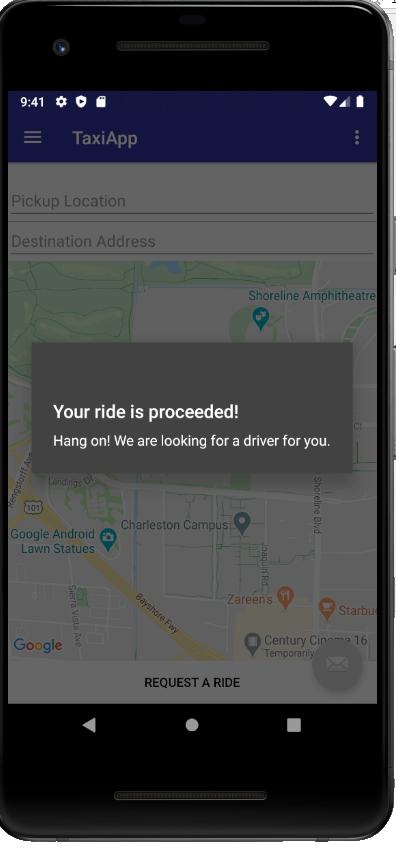
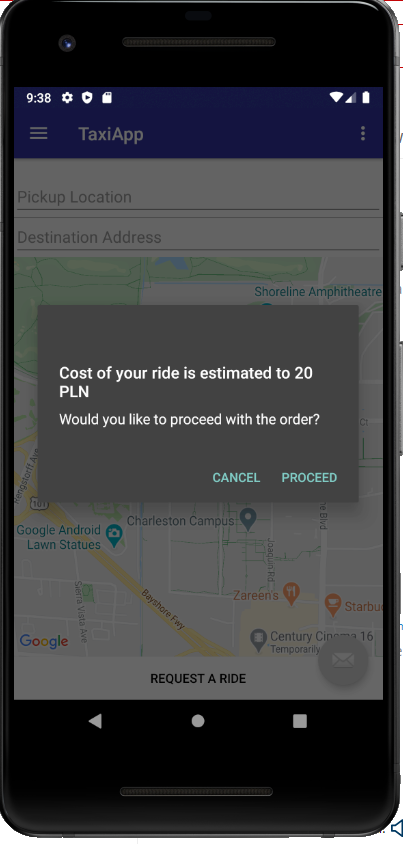
Rozpoczęto implementację logowania za pomocą konta Google. Aktualnie proces ten odbywa się wewnątrz aplikacji, chociaż w niektórych przypadkach powoduje to problemy z poprawnym zalogowaniem. W następnych krokach implementacji opcja ta zostanie zmodyfikowana tak, aby do logowania wykorzystywana była przeglądarka internetowa. Jest to podejście zalecane przez Google.





1. **Aplikacja Android**

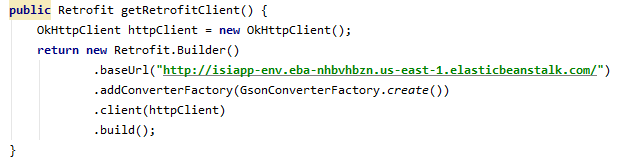
Zaimplementowano dialog informujący o obliczonym koszcie przejazdu, który umożliwia odrzucenie lub akceptację przejazdu. Po akceptacji przejazdu przez użytkownika musi zostać wysłane żądanie do serwera, który przydzieli kierowcę (komunikacja z serwerem zostanie zaimplementowane w przyszłym tygodniu).



### Postęp prac implementacyjnych - 18.05.2020

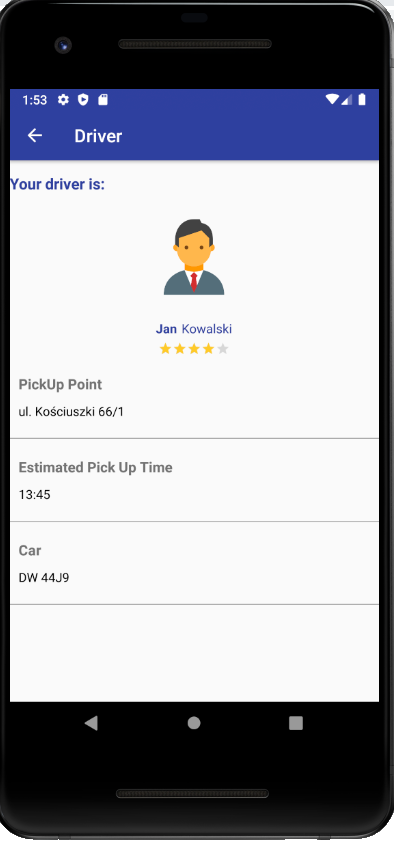
1. Android

Zaimplementowano komunikację z serwerem Spring Boot. wykorzystano w tym celu bibliotekę Retrofit.



Jak na razie wykorzystywane są endpoint /auth/login i /ride/getCost.

Ponadto dodano nową aktywność DriverActivity, która informuje użytkownika o przydzieleniu kierowcy do jego przejazdu i wyświetla podstawowe dane kierowcy i przejazdu.

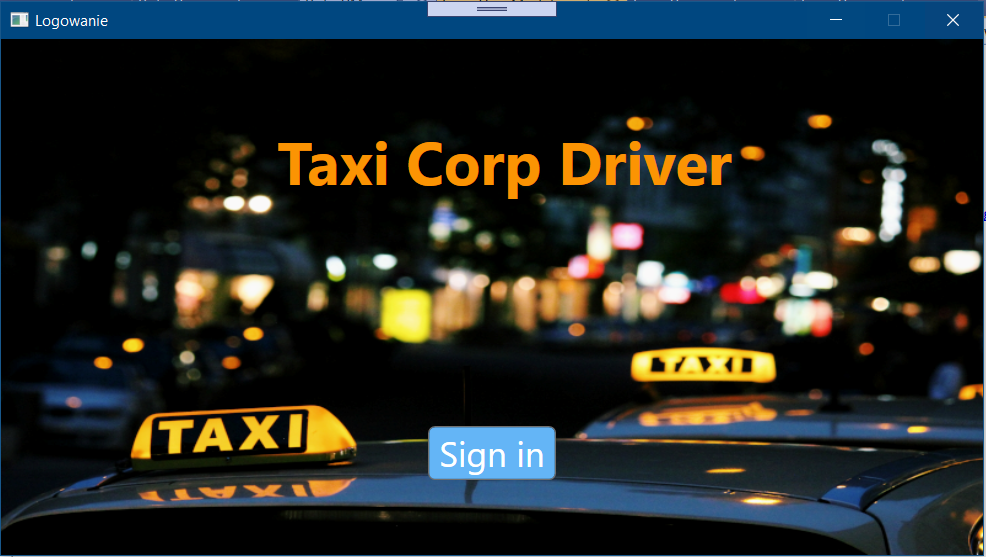


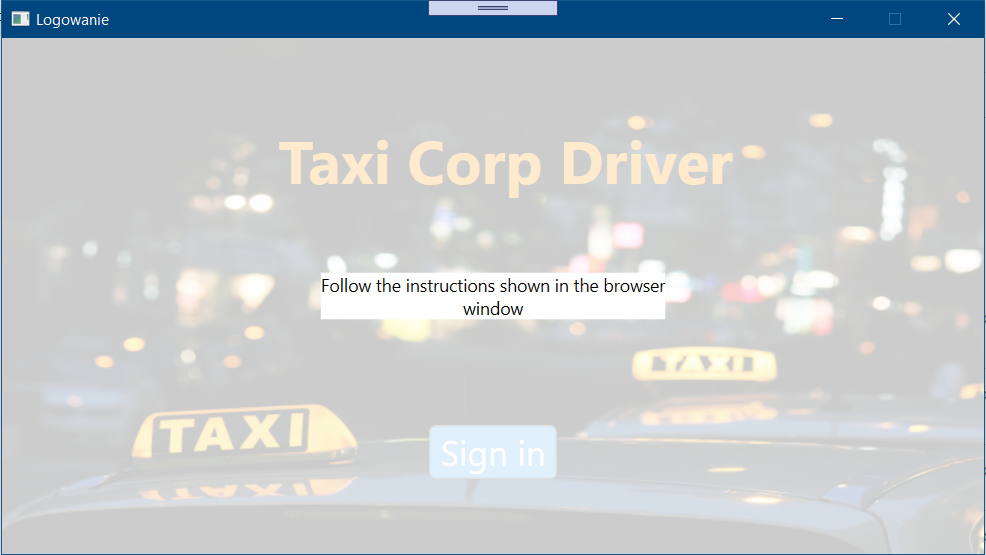
1. WPF

Zmodyfikowano sposób logowania do aplikacji kierowcy za pomocą konta Google, tak aby wykorzystana była zainstalowana przeglądarka internetowa, a nie komponent frameworka WPF - WebBrowser. Jest to podejście zalecane przez Google do integracji z aplikacjami napisanymi w WPF. Dzięki temu udało się rozwiązać występujące wcześniej problemy.

Oprócz tego zmodyfikowano i przetestowano integrację z serwerem, która została dotychczas zaimplementowana. Obecnie obejmuje ona logowanie oraz pobieranie ceny - endpointy /auth/login oraz /ride/getCost.

W toku są prace równiez nad designem aplikacji WPF, który zostanie uskuteczniony w przyszłym tygodniu.





1. Backend

Changelog v.18.05:

* Usuwanie usterek przy komunikacji z aplikacją Androidową
* Stworzenie kont dla autoryzacji do backendu dla aplikacji Android i WPF i dystrybucja danych do logowania w sposób chroniony
* Usuwanie usterek w logowaniu do backendu dla aplikacji WPF
* Zmiana formatu zapytania dla endpointu /ride/getCost

(Kod pozostał bez większych zmian, wersja kodu jest w poprzednim releasie)

### Postęp prac implementacyjnych - 24.05.2020

Android

Implementujemy ostatni krok procesu biznesowego “Zamawianie taksówki” (pierwszy BPMN) czyli płatnośc za przejazd. Zrobiliśmy integrację aplikacji Android z PayU. Zostały dodane już nowe layouty umożliwiające płatność. Jednak cały przepływ informacji - komunikacja z API PayU nie został jeszcze zrealizowany w całości.

### 

WPF

Zaprojektowany został interfejs obsługi przejazdów wraz z możliwością podjęcia przejazdu, oraz obliczaniem jego ceny na żądanie użytkownika. Przejazdy zostały też przygotowane do przyjęcia danych z endpointu /Rides, z którego będą pobierane dane o oczekujących przejazdach.

**Backend**

Changelog:

* Refaktoryzacja kodu i dodanie komentarzy
* Usunięcie z kodu niektórych części, w których korzystano z Java 8 streamów dla przyspieszenia działania części aplikacji, dla których szybkość działania jest ważna
* Przygotowanie planu implementacji części potrzebnych dla Androida i WPF

### Postęp prac implementacyjnych - 01.06.2020

#### Backend

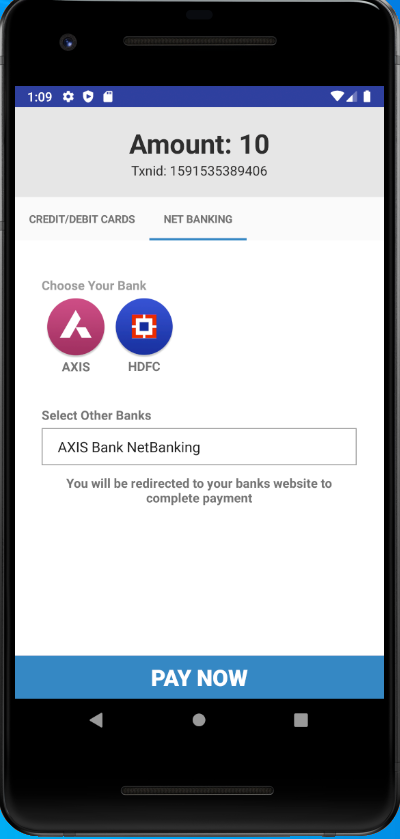
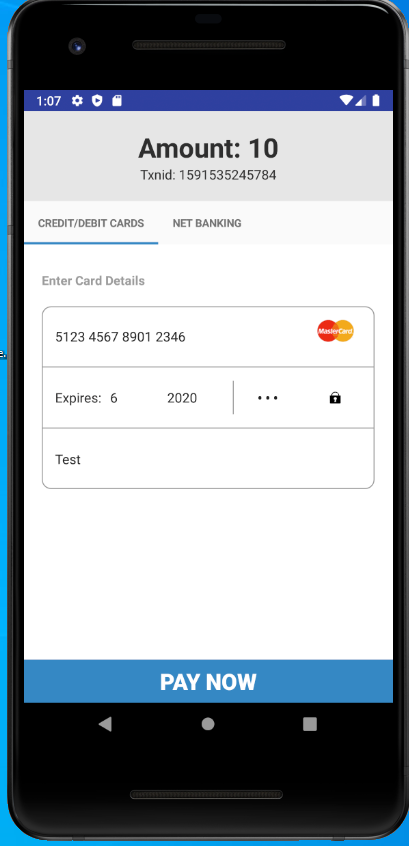
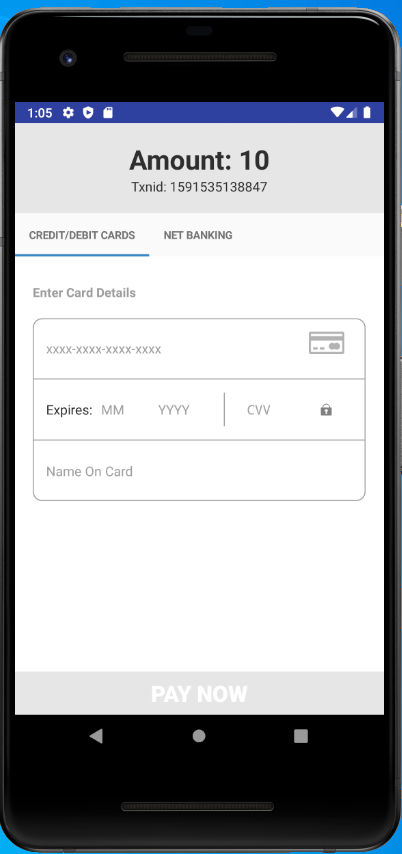
Changelog:

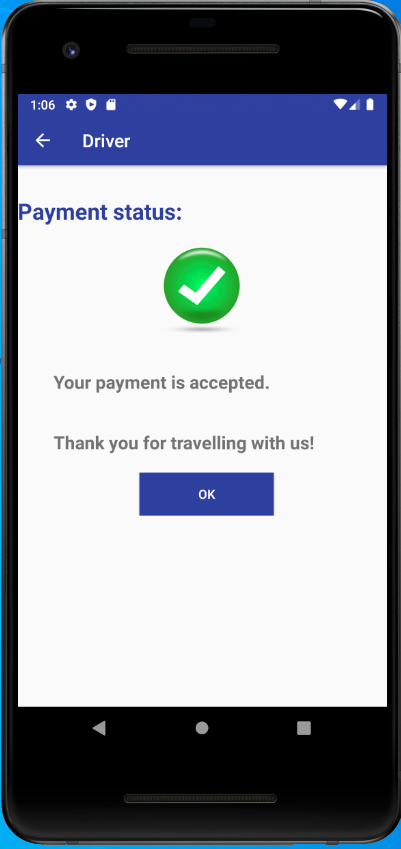
* Dodanie metod dla połączenia klienta z przejazdem
* Dodanie metod dla połączenia kierowcy z przejazdem
* Zmiana struktury obiektów i bazy danych dla ułatwienia przetwarzania informacji o przejazdach w aplikacji frontendowej

### Postęp prac implementacyjnych - 08.06.2020

#### Android

* Została zakończona integracja z bramką płatności PayU. W systemie jest możliwość płatności kartą i przez stronę banku. W ramach tej aplikacji są wykorzystywane środowisko testowe i testowe dane karty kredytowej, które są dostępne na stronie PayUMoney.
* Dokonanie płatności jest ostatnim elementem procesu biznesowego “Zamówienie przejazdu”





### Backend:

Changelog:

* Testowanie za pomocą różnych scenariuszy użycia
* Testowanie integracji z aplikacją Android
* Testowanie działania w trybie dwóch serwerów backendowych uruchomionych na maszynach lokalnych ze wspólną bazą danych