

**WARSZAWSKA
WYŻSZA SZKOŁA INFORMATYKI**

**PRACA DYPLOMOWA
STUDIA PIERWSZEGO STOPNIA**

Bartosz Rabij

Numer albumu: 9655

**Projekt i implementacja wybranych funkcji systemu
wspomagającego pracę krajowego transportu autobusowego**

**Promotor:
mgr inż. Andrzej Ptasznik**

Praca spełnia wymagania stawiane pracom dyplomowym na studiach pierwszego stopnia.

Spis treści

1. DZIEDZINA PROBLEMU	4
1.1. Zarys historyczny transportu autobusowego	5
1.2. Opis i porównanie istniejących rozwiązań	11
1.2.1. Veritum	11
1.2.2. Planbus.pl	12
1.2.3. optibus.com	13
1.3. Polski międzymiastowy transport autobusowy na przestrzeni kilku ostatnich lat....	14
1.4. Opis problemu.....	15
2. ANALIZA	17
2.1. Cel i zakres pracy.....	17
2.2. Cele biznesowe	18
2.3. Podstawowe założenia systemu wspierającego pracę transportu autobusowego	19
2.4. Specyfikacja wymagań funkcjonalnych systemu	21
2.4.1. Serwis internetowy dla pasażera.....	22
2.4.2. Aplikacja mobilna dla kierowcy	30
2.5. Specyfikacja wymagań pozafunkcjonalnych systemu.....	37
2.6. Modelowanie danych przy użyciu diagramu encji	39
2.7. Modelowanie czynności i zakresu odpowiedzialności systemu	42
2.7.1. Serwis internetowy dla pasażera.....	42
2.7.2. Aplikacja mobilna dla kierowcy	44
2.8. Bezpieczeństwo i ochrona danych.....	46
2.9. Wstępna architektura systemu	47
3. PROJEKT.....	49
3.1. Wybrane technologie	49
3.2. Projekt interfejsu użytkownika dla serwisu internetowego	52
3.2.1. Strona główna serwisu.....	52
3.2.2. Zakładka 1: Zakup biletu.....	53
3.2.3. Zakładka 2: Panel pasażera	54
3.2.4. Zakładka 3: Zaplanuj podróż.....	57
3.2.5. Przycisk 1: Logowanie, wylogowanie i rejestracja	58
3.3. Projekt interfejsu użytkownika dla aplikacji mobilnej	59
3.3.1. Logowanie i zgłoszanie problemu	59
3.3.2. Zakładka 1: Trasa	61
3.3.3. Zakładka 2: Mapa autobusu.....	61

3.3.4.	Zakładka 3: Lista przystanków.....	62
3.3.5.	Zakładka 4: Kasowanie biletu	63
3.4.	Projekt bazy danych.....	64
3.5.	Architektura systemu	66
4.	IMPLEMENTACJA	69
4.1.	Implementacja bazy danych.....	71
4.1.1.	Omówienie sposobu implementacji bazy danych metodą „code first”	71
4.1.2.	Utworzenie i mapowanie klas jako gotowych tabel w bazie danych	72
4.2.	Implementacja serwisu internetowego.....	80
4.2.1.	Wykorzystane wzorce i podejścia do stworzenia aplikacji przeglądarkowej	80
4.2.2.	Strona główna serwisu.....	85
4.2.3.	Logowanie i rejestracja.....	86
4.2.4.	Zakup biletu	88
4.2.5.	Panel klienta	92
4.2.6.	Zaplanuj podróż	94
4.3.	Implementacja aplikacji mobilnej.....	95
4.3.1.	Wykorzystane wzorce i podejścia do stworzenia aplikacji mobilnej	95
4.3.2.	Logowanie i zgłaszanie problemu	96
4.3.3.	Widok aktualnej trasy	98
4.3.4.	Mapa autobusu.....	98
4.3.5.	Lista przystanków	99
4.3.6.	Skanowanie biletów.....	101
5.	PODSUMOWANIE I WNIOSKI.....	104
	Bibliografia.....	106

1. DZIEDZINA PROBLEMU

Pierwszy rozdział pracy inżynierskiej omówi i przybliży transport autobusowy. W celu lepszego zrozumienia kontekstu powstawania systemu wspierającego określona branżę, pierwszym poruszonym zagadnieniem jest przedstawienie tego, jak kształtował i rozwijał się samochodowy, zbiorowy transport ludności.

Zapoznanie się z istniejącymi i działającymi na rynku rozwiązaniami, czy to polskimi, czy zagranicznymi, pozwoli skupić się na brakach w oprogramowaniu w tejże branży, czy ulepszeniu dotychczasowych rozwiązań. Pomoże to też, osobom na codzienna nie zajmującym się branżą przewozów autobusowych, zrozumieć zasadę działania takiego oprogramowania. Jest to przedmiotem rozważań w drugim podrozdziale.

Kluczową uwagę skupiono na polskim międzymiastowym transporcie autobusowym. Jego aktualną sytuację omawia trzeci podrozdział. Zostały opisane problemy i wyzwania wytyczone na poziomie rządowym. Zidentyfikowane problemy stanowią istotny kontekst dla dalszej części pracy, a szczegółowa analiza tego obszaru umożliwi lepsze zrozumienie kontekstu projektowanego systemu wspomagającego transport autobusowy.

Ostatni, czwarty podrozdział, przedstawia potencjalny problem, z jakim mogą się borykać firmy zajmujące się transportem autobusowym. Opis ten powstał na podstawie analizy źródeł w postaci lokalnych serwisów informacyjnych i wymianie informacji pomiędzy działającymi przewoźnikami na terenie kraju.

1.1. Zarys historyczny transportu autobusowego

Wraz z wynalezieniem koła w epoce kamienia, w okolicach 3500 roku przed naszą erą, ludzkość zapoczątkowała rozwój dzisiejszego transportu autobusowego. W czasach pradawnych poruszano się głównie po lądzie używając siły własnych mięśni, przygotowując wcześniej udogodnienia pomagające w przemierzaniu drogi pieszo. Używano wtedy na przykład czegoś, co jest wykorzystywane aż do dzisiaj w przemieszczaniu się po obszarach ośnieżonych, pierwszych rakiet śnieżnych mających zamocowane kolce na drewnianej bazie wynalezku przytwierdzanego pasami zwierzęcej skóry do stopy.



Rysunek 1-1 Drewniany wózek transportowy (źródło: <https://razib.substack.com/p/steppe-10-going-nomad.>)

Okolice roku 3000 przed naszą erą historycy uznają za okres pierwszego podejścia do transportu drogowego. Skonstruowano wtedy prymitywny dwukołowy wózek. Takie wózki były konserwowane. Koła mocowano, do wcześniej przymocowanych na stałe mniejszych bali pełniących funkcję osi, uprzednio je smarując tłuszczami zwierzęcymi lub roślinnymi, co zmniejszało tarcie między elementami ruchomymi powstałego wózka. Próbowano wtedy też przemierzać małe zbiorniki wodne. Ludność dowiedziała się, że drzewa i kłody unoszą się na wodzie, odpowiednio je przygotowywała do krótkiego rejsu. Drażono wnętrze drzewa tak, by umożliwiało ono siedzenie w nim. Takie rozwiązanie zapoczątkowało wykorzystywanie zbiorników wodnych jako formy transportu dla ówczesnej ludności. [1]

Z biegiem lat technologia transportu lądowego się rozwijała. W około 2500 roku przed naszą erą zaczęły powstawać ciężkie, czterokołowe pojazdy wykorzystywane do przewozu bydła. Pół wieku po tych wydarzeniach opracowano pierwszą uprząż zapoczątkowującą transport konny. Od tego czasu postęp w dziedzinę transportu lądowego zaczął nabierać dużego rozpędu.



Rysunek 1-2 Powóz konny, wynalazek Blaise Pascala (źródło: <http://lantanews.blogspot.com/2012/05/history-of-bus.html>)

Pierwszą osobą, która zaproponowała rozwiązywanie problemów z systemu transportu publicznego był francuski inżynier i matematyk Blaise Pascal. W Paryżu roku 1662 uruchomił kilka tras powozów konnych. Początkowo odniosły one niemały sukces, jednak dosyć szybko ich popularność zaczynała maleć. Wspomnianymi powozami mogła podróżować jedynie francuska szlachta, a ludność z niższych warstw społecznych nie miała nawet szansy, by przekonać się, co oferuje i jak działa pomysł Pascala. Ostatecznie pierwszy system transportu publicznego okazał się rozwiązańiem nietrafionym. Musiało minąć kolejne 150 lat, by ktoś zorganizował i spopularyzował drogowy transport pasażerski. [2]

W przeciwieństwie do poprzedniego rozwiązania, następne nadchodzące udogodnienie służyło ludziom zdecydowanie dłużej. Przełom miał miejsce ponownie we Francji. W 1828 roku powstał omnibus, zaprzęgnięty najczęściej w trzy konie pojazd. Mógł on przewodzić nawet do 42 pasażerów jednocześnie. Podobną linię omnibusa, w tym samym roku co Francuzi, wdrożono również w Nowym Jorku, a następnie w wielu innych miastach Stanów Zjednoczonych. Mimo rewolucyjnego, jak na tamte czasy rozwiązania, miało ono kilka

poważnych wad. Ówczesne drogi, w najlepszym wypadku, miały jedynie utwardzoną nawierzchnię kostką brukową, przez co były bardzo wyboiste. Siedzenia omnibusów nie miały zawieszenia, miękkich obić czy wyściółek na siedzeniach. Miało to negatywny wpływ na komfortu jazdy pasażerów podczas dłuższych podróży. Wysokie ceny biletów też nie zachęcały do korzystania ze świeżego wynalazku, odstraszały mniej zamożnych obywateli. Tym razem nie było rozgraniczeń na to, kto mógł korzystać z omnibusów. Były one dla wszystkich, których było na nie stać. Nowopowstała idea transportu zbiorowego bardzo przypadła do gustu ówczesnej klasie średniej. Ludność wywodząca się z niej nie mogła jeszcze sobie pozwolić na samodzielnny zakup środka transportu tego typu. Chcąc uniknąć długiego i wyczerpującego spaceru, to właśnie rozrastająca się wtedy klasa średnia korzystała najczęściej i najwięcej z sieci omnibusów. Taka kombinacja czynników pozwoliła na utrzymywanie się środka transportu, a w dalszej perspektywie ich postępujący rozwój i popularyzację. [3]



Rysunek 1-3 Szynowy omnibus z zaprzęgiem konnym (źródło: <https://www.historyanswers.co.uk/inventions/what-were-horsecars/>)

Wartą nadmienienia jest etymologia angielskiego słowa „bus”. Jego dzisiejsze znaczenie zapoczątkował wcześniej wspomniany „omnibus”, jest jego krótszą formą używaną w czasach popularności tego środka transportu. Zaś samo słowo „omnibus”, jak twierdzą językoznawcy, łączy się z łacińskim słowem „multos” – wielu, wszyscy lub z nazwą jednego z francuskich linii omnibusowych „Omnes”.

Schyłek wieku XIX dla rozwoju przewozu osób był dosyć kluczowym. Poczyniono wtedy wyraźne ulepszenia koncepcji omnibusów. Zaczęto rozkładać torowiska na wcześniej powytyczanych trasach, co umożliwiło znacznie płynniejszą i wygodniejszą podróż dla nawet trzykrotnie większej liczby pasażerów. W XIX-wiecznych Stanach Zjednoczonych istniało łącznie około 50 000 km torowisk przystosowanych do poruszania się po nich pasażerskich zaprzęgów konnych. Wykorzystywanie koni do tego typu prac zradzało obawy społeczne. Zwierzęta mogły pracować jedynie przez około dwie godziny, więc żeby zapewnić odpowiednią systematyczność i częstotliwość przejazdów dla jednej tylko linii wykorzystywano nawet kilkanaście koni. Narastające koszty obsługi połączeń i obawy społeczne jakie towarzyszyły dla tego typu przedsięwzięć inspirowały ówcześnie środowiska inżynierów i konstruktorów do ciągłej pracy nad rozwojem i podejmowania prób porzucenia poruszających się po torach zaprzęgów konnych na rzecz pukającej do drzwi technologii i automatyki. [4]



Rysunek 1-4 Wagon kolej linowej. (źródło: <https://www.sfmta.com/getting-around/muni/cable-cars/cable-car-history>)

Pewien Amerykanin mieszkający w San Francisco, Andrew Smith Hallide stworzył pierwszą na świecie kolej linową poruszającą się po ulicach miasta. Zasada jej działania polegała na puszczeniu pomiędzy szynami torowiska poruszającej się stalowej liny, do której umocowane były odpowiednie zaczepy, a do nich już wagony, wcześniej ciągnięte przez konie. Nowopowstałe rozwiązanie nie wymagało już udziału zwierząt w organizacji przewozu osób, choć nadal funkcjonowały omnibusowe linie.

Pod koniec XIX wieku linowa kolej szynowa stopniowo przemieniała się w trolejbusy, później nazywane tramwajami i ostatecznie zastąpiły używane jeszcze wtedy wagony konne zamykając erę omnibusów napędzanych przez zwierzęta. Poruszały się po szynach, energię czerpały z sieci trakcyjnej podwieszonej nad torowiskami. Tramwaje miały gotową infrastrukturę, poruszały się na wcześniej skonstruowanych szynach dla omnibusów. Po

istniejących trasach podróżowało więcej ludzi na większe odległości po niższych kosztach. Trolejbusy uznawane były za jeden z najbardziej wpływowych wynalazków. Tramwaje sprawiły, że codzienna podróż stała się ogólnodostępna. Poskutkowało to przenoszeniem się mieszkańców gęsto zamieszkanych centrów dużych miast na jego przedmieścia kształtuając założki dzisiejszych wielomilionowych metropolii. [5]



Rysunek 1-5 Pierwszy autobus Karla Benza (źródło: https://www.mercedes-benz-bus.com/pl_PL/brand/omnibus-magazin/125-years-buses.html)

Współcześnie znane autobusy mają swój początek, równolegle z samochodami osobowymi, w roku 1895, kiedy to Karl Benz, niemiecki inżynier, konstruuje niewielki autobus z napędem silnikowym wyglądem przypominający nieco wcześniejsze omnibusy. Pojazd zwany „Landauer” wyposażono w silnik z pojedynczym cylindrem o mocy 5 koni mechanicznych. Dystans 15 km został przez pojazd pokonany w 1 godzinę i 20 minut. Problem stwarzały bardziej strome wzniesienia, kiedy to pasażerowie zmuszeni byli wysiąść z pojazdu i go popchać. Zaczęły powstawać większe pojazdy. W roku 1905 niemiecka firma Daimler znacznie usprawnia koncepcję Carla Benza i proponuje swoje rozwiązanie. Ich pojazd może pomieścić zdecydowanie więcej pasażerów niż wcześniejszy „Landauer”, ma mocniejszy silnik – 28 KM. W 1911 roku niemiecka poczta używa łącznie 164 autobusy, z czego większość stanowiły pojazdy firmy Daimler. Pierwszym masowo produkowanym autobusem był dwupiętrowy „B-type” zaprojektowany przez głównego inżyniera londyńskiego

przedsiębiorstwa General Omnibus Company. Z wyglądu bardzo przypominał niemieckiego Landauera. Do końca 1920 roku zbudowano prawie 3000 sztuk tego pojazdu, z czego kilkaset uczestniczyło w zachodnim froncie pierwszej wojny światowej.



Rysunek 1-6 Autobus „Landauer” firmy Daimler (źródło: <https://www.themotormuseuminminiature.co.uk/inv-karl-benz.php>)

Wczesne zmotoryzowane dwudziestowieczne autobusy składały się z podwozia wcześniej zmodyfikowanej ciężarówki i otwartego dachu. W Stanach Zjednoczonych głównym producentem była firma Yellow Coach Manufacturing. Założona została w 1923 roku przez Johna D. Hertza. Opracowano i wyprodukowano wtedy pierwszy pojazd z nadwoziem bardzo przypominającym dzisiejsze autobusy. Dwa lata później firmę przejęło General Motors i zmieniło nazwę na Yellow Truck and Coach Manufacturing Company. W ciągu kolejnych 30 lat rozwoju autobusy wyposażono w silnik umieszczony z tyłu i większy rozstaw kół. Kilka lat później zaczęto produkcję pojazdów napędzanych olejem napędowym.

Wraz z pojawianiem się większej liczby samochodów, zmniejszaniu ich kosztów ich produkcji i kupna, ilość chętnych na podróżowanie publicznym transportem malała. W latach 60-tych i 70-tych kształtowała powszechność posiadania własnego samochodu. Pierwsza połowa lat 50-tych przyniosła udoskonalenie w postaci pneumatycznego zawieszenia umożliwiającego trzymanie stałego prześwitu autobusu niezależnie od ilości pasażerów na pokładzie. [6]

Ze względu na, między innymi, pogarszające się warunki środowiska transport powoli zaczyna przechodzić kolejną znaczącą zmianę. Wraz z pojawieniem się i ciągłym zdobywaniem popularności przez elektromobilność, co raz częściej można dostrzec na ulicach

w pełni elektryczne pojazdy transportu publicznego. W Stanach Zjednoczonych, do 2045 około 40% wszystkich autobusów będzie elektryczne. Obecnie panują obawy co do ich zasięgu i ceny, elektryczne odpowiedniki spalinowych pojazdów potrafią kosztować nawet kilkukrotnie więcej, co zniechęca podmioty świadczące usługi transportu autobusowego do przejścia na model bardziej ekologiczny.

1.2. Opis i porównanie istniejących rozwiązań

By móc zapoznać się z wymaganiami, jakie oczekuje rynek od budowanego oprogramowania, warto zapoznać się z istniejącymi rozwiązaniami, podobnymi do tego, co będzie planowane do budowy. W tym rozdzielne omówiono kilka rozwiązań działających na roku, trzy z nich będą rozwiązaniami polskimi, jedno, zagraniczne.

1.2.1. Veritum

Zbiór rozwiązań i systemów ERP¹ do organizacji pracy mający kilka modłów:

- Komunikacja Miejska i Gminna
- Transport Osobowy
- Transport Towarowy

Biorąc pod uwagę tematykę pracy inżynierskiej warto przybliżyć „Moduł Transport Osobowy”. Jego funkcjonalność pozwala:

- zarządzać taborem autobusów,
- tworzyć rozkład jazdy,
- planować grafik i realizować usługi przewozowe,
- zbiorczo generować karty drogowe i sczytywać tankowania paliwowe,
- rozliczać nadgodziny, czas pracy i wynagrodzenia kierowców.

Veritum Transport Osobowy może być wykorzystywany przez przewoźników organizujących trasy pasażerskie na szczeblu międzymiastowym. System w całości integruje całość procesów dotyczących planowania, harmonogramów i rozliczana przewozów w przedsiębiorstwie zajmującym się transportem autobusowym. Samodzielnie potrafi wygenerować rozkład jazdy, czy zimportować już istniejące.

¹ ERP - Enterprise Resource Planning, planowanie zasobów przedsiębiorstwa. Typ oprogramowania wspomagający zażądanie zasobami przedsiębiorstwa we wszystkich jego działach (płace, kadry, księgowość, transport, magazyn itp.).

Wykorzystując odpowiednie algorytmy istnieje możliwość przygotowywania grafików, planów dla kierowców i pojazdów będących na wykorzystaniu. Kursy mogą być optymalizowane pod względem ich przebiegu czasowego.



Rysunek 1-7 Widok strony veritum.pl (źródło: <https://www.veritum.pl/dedykowane-rozwiazania-erp/program-do-zarzadzania-transportem-osobowym/>)

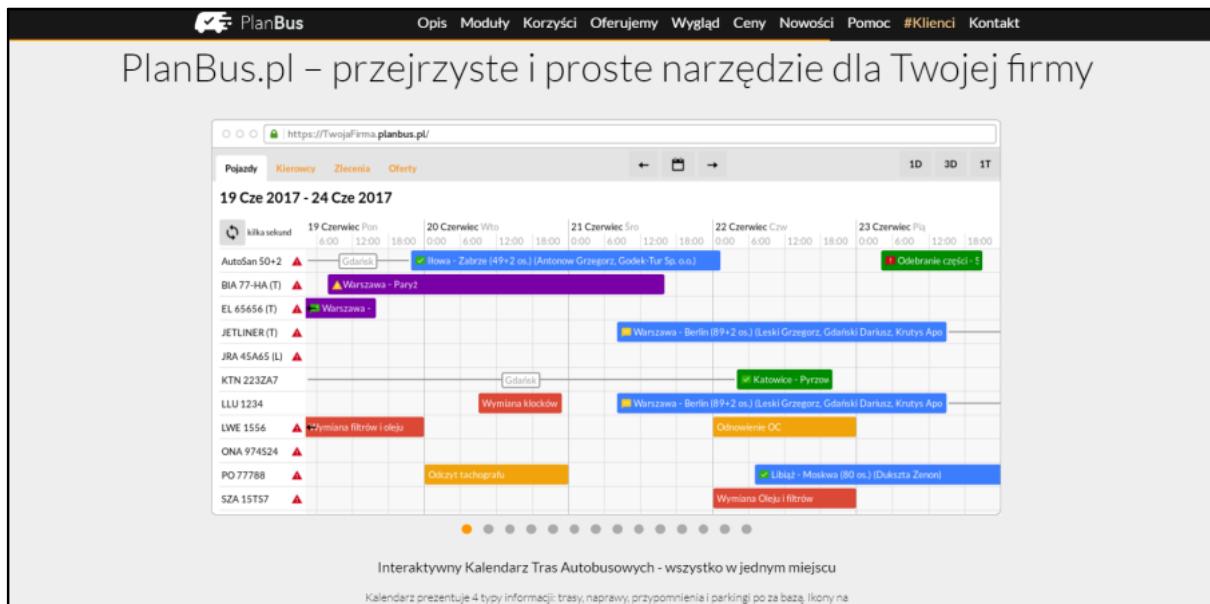
Rozwiązanie również zapewnia wsparcie dla osób pełniących rolę dyspozytorów. Kontrolom poprzez wspominany system podlegają czas pracy kierowców, dostępność i wykorzystanie pojazdów, czy zgłaszone usterki. Po przygotowaniu odpowiedniego modułu oprogramowania, tworzone również mogą być raporty dotyczące rozliczeń za paliwo, koszt paliwa na daną trasę i inne tematy podobne.

Veritum jest niemałym systemem ERP oferującym mnogość rozwiązań, od modułów bezpośrednio związanych z księgowością, aż po te, odpowiadające za transport autobusowy, czy też towarowy. Zakup rozwiązania jest jednorazowy, wybierane są moduły, które interesują klienta i na podstawie wymagań prezentowana jest spersonalizowana oferta. Każdy z modułów można łączyć z pozostałymi, będącymi w obrębie systemu Veritum. [7]

1.2.2. Planbus.pl

PlanBus jest jednym z systemów informatycznych służący zarządzaniu usługami wynajmu autobusów i przewozem osób. Całe rozwiązanie jest aplikacją działającą jedynie w przeglądarce. Bazuje na kalendarzu, w którym prezentowane są wszystkie najważniejsze dane

na osi czasu. Z poziomu aplikacji przeglądarkowej ma się dostęp do kilku modułów: Klienci, Oferty, Zlecenia, Pojazdy i Kierowcy. Każdy z nich jest dostępny pod inną zakładką w oknie kalendarza.



Rysunek 1-8 Widok strony planbus.pl (źródło: <https://www.planbus.pl/>)

To rozwiązanie oferuje stały dostęp do systemu, z każdego miejsca na ziemi i urządzenia wyposażonego w przeglądarkę internetową. Możliwa jest praca na wielu stanowiskach, menedżer, dyspozytor czy logistyk mają odrębne uprawnienia w systemie i dostęp do różnych funkcjonalności systemu.

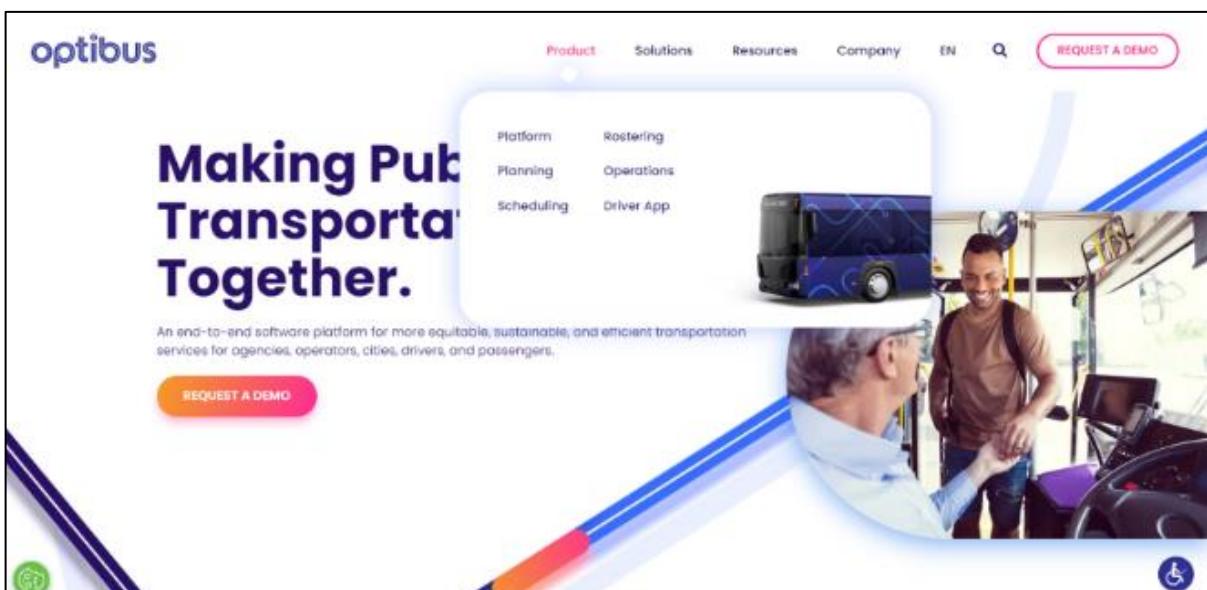
Korzystanie z usługi opiera się na kilku progach abonamentu dostosowanego do ilości pojazdów w przedsiębiorstwie. Cennik zaczyna się od 49,90 zł netto za każdy pojazd miesięcznie za nie mniej niż 10 pojazdów, kończy na 29,90 zł miesięcznie / pojazd za maksimum 100 pojazdów będących w użyciu w przedsiębiorstwie. [8]

1.2.3. optibus.com

Jedno z popularniejszych zagranicznych rozwiązań do organizacji transportu autobusowego. Oprogramowanie oparte jest na rozwiązaniach chmurowych. Obejmuje moduły odpowiedzialne za planowanie i harmonogramowanie, monitorowanie działań w zajezdni, monitorowanie i wymianę informacji z kierowcami. Oprogramowanie składa się ze współpracujących ze sobą narzędzi:

- Planowanie – zawiera moduły odpowiedzialne za planowanie strategiczne i operacyjne, co pozwala łatwo i szybko planować, analizować i optymalizować całą sieć transportową.

- Harmonogramowanie – narzędzie oparte o algorytmy optymalizacyjne i uczenie maszynowe. Zbierane są dane dotyczące ruchu każdego kierowcy i pojazdu, na podstawie nich wypracowywane są najbardziej wydajne i zoptymalizowane rozkłady przejazdów.
- Tworzenie grafików – minimalizuje ręczne ustalanie grafików dla kierowców. Ustala schematy dni wolnych, nadgodzin, gwarantowanego czasu odpoczynku między zmianami i inne.
- Operacyjność – śledzenie kierowców w czasie rzeczywistym i kontakt z nimi za pomocą systemu alokacji pojazdów.
- Aplikację dla kierowcy - zapewnia kierowcom łatwy dostęp do wszystkich potrzebnych informacji. Przed rozpoczęciem kursu kierowca może przejrzeć trasę, zgłosić swoje uwagi do niej.



Rysunek 1-9 Widok strony optibus.com (źródło: <https://www.optibus.com/>)

Oprogramowanie Optibus wydaje się być tym najbardziej kompletnym. Działa w chmurze, więc jest użyteczne wszędzie tam, gdzie istnieje dostęp do sieci. Wykorzystane technologie uczenia maszynowego i różne algorytmy znaczco przyspieszają organizację i planowanie dla taborów autobusów. [9]

1.3. Polski międzymiastowy transport autobusowy na przestrzeni kilku ostatnich lat

Obecny rynek polskich, dalekobieżnych przewozów autobusowych stale się kurczy. Powodów tego jest kilka. Rynek jest słabo zorganizowany, niedawna pandemia COVID-19 miała dosyć silny wpływ na pogorszenie się tego odłamu branży przewozów samochodowych,

pomimo dużych inwestycji w infrastrukturę drogową naszego kraju. W latach 2016 – 2021 liczba pasażerów korzystających z usług transportu dalekobieżnego oferowanego przez firmy zatrudniające więcej niż 9 osób spadła z 478,6 mln do 314 mln. Wynikiem spadków było zakończenie działalności przez wielu drobniejszych prywatnych przewoźników, a kolejne przedsiębiorstwa wywodzące się jeszcze z PKS zostały postawione w stan likwidacji. Problem upadania transportu w dużym stopniu dotyczy obszary pozamiejskie, przez co rośnie wykluczenie transportowe tych obszarów.

Obecne przewidywania rozwoju rynku komunikacji autobusowej się pogarszają. Szacuje się, że poprawa może nadjść nie wcześniej niż w 2025 roku. Przedsiębiorstwa otrząśające się po prawie trzech latach wyraźnych spadków, rzadko wymieniają starsze pojazdy i czynią inne inwestycje. Na poprawę stanu nie wpływają też rosące ceny biletów. Ciężko jest zaoferować możliwości zakupu biletu za niższą cenę.

Łączna długość wytyczonych tras krajowych linii komunikacji autobusowej zmalała o ponad połowę, z około 1 miliona kilometrów w roku 2010 do około 470 tysięcy kilometrów w roku 2022. Według danych GUS, w tym samym okresie, liczba przejechanych kilometrów na trasach autobusowych wynosiła 1,02 miliarda i 418 milionów. Warto wspomnieć, że największy spadek nastąpił w roku 2019 i 2020. Tendencja powoli wyhamowuje, co może pomóc w postępującym wykluczeniu komunikacyjnym na terenach wiejskich. Rząd od 2021 roku przeznaczył na program przywracania przewozów autobusowych 800 mln zł, z czego, do połowy 2022 roku, udało się wydać jedynie okolice 9% tej kwoty. [10] [11]

1.4. Opis problemu

Firma X uzyskał dofinansowanie z Funduszy Europejskich postanowiła zamówić oprogramowanie specjalistyczne dla swoich kierowców. Jako że owa firma stara się rozwijać, z małych, podmiejskich, potem międzymiastowych kursów na te ponadmiastowe, międzywojewódzkie, ogólnopolskie. Ale najpierw musi pewne procesy zautomatyzować, przejść na cyfrowy model zarządzania działalnością.

Obecnie kierowcy w sposób manualny „kasują” bilety. Notorycznie zdarzają się „przeładowania” autobusów, bo ciężko zapanować nad 40-kilku osobowym tłumem w pojeździe. Zdarzało się, że kierowca ze świeżą licencją zabłędził na trasie, jedyne co miał, to odręczny spis przystanków i orientacyjne godziny przyjazdu i odjazdu, jakie go obowiązywały na aktualnej trasie. O sprawach dotyczących informowania pasażerów o opóźnieniach na trasie obecnie również nie ma mowy.

Bardzo często zawodzi komunikacja pomiędzy kierowcami a koordynatorem. Reakcje na wydarzenia na trasie docierają, jeżeli w ogóle, do kierowców z bardzo dużym opóźnieniem, a ci nie mają jak na nie odpowiedni zareagować.

Przedsiębiorstwo musi zmierzyć się z rosnącą konkurencją ze strony innych, większych i mniejszych przewoźników. Nie są prowadzone żadne akcje marketingowe mediach społecznościowych i Internecie, gdzie większość potencjalnych klientów szuka informacji o połączeniach i dokonuje zakupu biletów.

W wyniku takich ilości niedopatrzeń pasażerów firmy wchłania konkurencja, co za tym idzie, spadają dochody. Potrzebne są rozwiązania unowocześniające branżę, w szczególności mniejszych przedsiębiorców i będące za razem łatwo dostępne i szybkie w implementacji i przyjazne w stosowaniu na co dzień.

2. ANALIZA

Czynności dotyczące analizy w procesie tworzenia systemu informatycznego dotyczą dokładnego zapoznania się z tematem projektowym. Analizuje się i określa wymagania co do funkcjonalności systemu, czy szacuje się koszty całego projektu. Dobrze wykonana analiza jest fundamentem dla dalszych czynności związanych z budowaniem systemu informatycznego i decydowaniu o jego kształcie.

Pierwszy z podrozdziałów precyzyjnie określa główne cele przyświecające pracy inżynierskiej oraz zakres czynności, jakie będą wykonane by ten cel osiągnąć.

Następnie, w kontekście budowanej aplikacji, określone są cele biznesowe, które przedsiębiorstwo będzie chciało osiągnąć poprzez stworzenie, wdrożenie i używanie systemu informatycznego.

Trzeci podrozdział objaśnia wstępne założenia dla rozwiązań skupiającego się na wsparciu transportu autobusowego. Definiowane są role w systemie oraz ogólna funkcjonalność.

Kolejno wyspecyfikowane są wymagania funkcjonalne i pozafunkcjonalne. Dla każdej z dwóch budowanych aplikacji w ramach całego systemu funkcjonalności zostały ujęte i przedstawione na diagramach przypadków użycia. Każdy z nich szczegółowo, wraz z przebiegiem głównym i przebiegami iteratywnymi opisano w tabeli.

Mając wyspecyfikowaną funkcjonalność całego systemu, w kolejnej części drugiego rozdziału, poświęconemu modelowaniu dynamicznych aspektów systemu na diagramach aktywności, ukazano dwie części systemu i kroki, jakie należy podjąć, by doprowadzić do żądanego rezultatu, jak na przykład zakup biletu.

Podrozdział „bezpieczeństwo i ochrona danych” rysuje obraz technik podejść potencjalnie wykorzystywanych w budowanym systemie do zapewnienia spójności i możliwe jak największego poziomu zabezpieczenia danych użytkowników.

Ostatni podrozdział przedstawi wstępную architekturę systemu. Uwzględnia początkowe, planowane podejścia, jakie będą mogły być wykorzystane podczas projektu, czy później implementacji systemu.

2.1. Cel i zakres pracy

Przy projektowaniu i implementacji systemu informatycznego dla krajowego transportu autobusowego, należy wziąć pod uwagę jego efektywność i komfort użytkowania. Praca ma na opracowanie projektu i implementację wybranych funkcjonalności takiego rozwiązania.

Czynnościami wyjściowymi przy tworzeniu systemu informatycznego dla krajowego transportu autobusowego było zebranie informacji na temat tej branży w Polsce. Wynikiem zebranych informacji będzie określenie celów biznesowych. Następnie powstanie szczegółowa specyfikacja wymagań funkcjonalnych, opisujących funkcjonalność systemu, oraz niefunkcjonalnych przedstawiających zewnętrzne czynniki wpływające na działanie całego systemu.

Po analizie wymagań nadaje się czas na zaprojektowanie całego rozwiązania. Składać się ono będzie z serwisu internetowego dla klienta - pasażera oraz aplikacji mobilnej dla kierowcy używanej w trakcie realizacji kursu. Przedstawionym zostanie również wstępny projekt interfejsu użytkownika dla obu elementów całego systemu. Oba moduły będą korzystały ze wspólnej bazy danych.

Jedną z końcowych faz przedstawionych w pracy będzie implementacja. Na podstawie wcześniej wypracowanego projektu powstaną wybrane moduły systemu informatycznego. Całość zostanie stworzona we wcześniej odpowiednio dobranych technologiach. Na tym etapie możliwe jest wystąpienie niewielkich zmian względem tego, co zostało przedstawione wcześniej w projekcie, by dostarczyć możliwie jak najlepsze i najbardziej przyjazne dla użytkowników rozwiązanie informatyczne.

2.2. Cele biznesowe

Z przedstawionego na końcu poprzedniego rozdziału opisu problemu klarują się następujące cele biznesowe przyczyniające się do efektywniejszego funkcjonowania przedsiębiorstwa zajmującego się transportem dalekobieżnym:

1. Oferowanie usług o największej wygodzie i dostępności: użytkownik będzie mógł zaplanować wcześniej podróż, zapłacić za nią online, interfejs będzie dostępny, przyjazny i intuicyjny dla ludzi z zaburzeniami widzenia.
2. Digitalizacja obiegu dokumentów uprawniających do jazdy: przejście z tradycyjnych, papierowych uprawnień do przejazdu na rzecz cyfrowych, generowanych przez system.
3. Zapewnienie punktualności przejazdów: system ma dostarczyć klientom usługi transportowe zgodne z aktualnymi rozkładami jazdy poprzez: zoptymalizowane trasy, reagowanie na problemy, aktualizowanie na bieżąco rozkładów jazdy zgodnie z zapotrzebowaniem.
4. Zwiększenie bezpieczeństwa przejazdu: strona główna serwisu internetowego na bieżąco będzie pokazywać aktualne powiadomienia i ostrzeżenia dotyczące korzystania z oferty przewoźnika, instrukcje postępowania w trakcie awarii.

Kontrolowana, z poziomu aplikacji mobilnej dla kierowcy, będzie również ilość wolnych i zajętych miejsc w pojeździe, co wyeliminuje problem pasażerów podróżujących na stojąco.

5. Pozyskanie nowych pasażerów: poprzez prostotę i wygodę użytkowania systemu, oraz łatwy dostęp do informacji pozwoli na powiększenie bazy swoich klientów.
6. Pozbycie się problemów komunikacyjnych wewnętrz przedsiębiorstwa: skoncentrowanie się na usprawnieniu komunikacji i wymianie informacji między różnymi działami (kierowca – koordynator), ponadto system umożliwia zautomatyzowanie wielu procesów, takich jak generowanie biletów.
7. Zwiększenie dochodów firmy: system zaoferuje łatwe i wygodne metody rezerwacji biletów oraz płatności online. Proces rezerwacji i płatności będzie intuicyjny i bezproblemowy, co przełoży się na zwiększoną liczbę dokonywanych transakcji i wzrost dochodów.

Realizacja powyższych celów biznesowych pozwoli na zwiększenie kontroli danego przedsiębiorstwa nad organizacją transportu, zoptymalizuje, zautomatyzuje i przyspieszy procesy obsługi klienta i wyeliminuje dotychczasowe błędy i niedogodności. Odpowiednio zidentyfikowane, zrozumiane i sformułowane cele biznesowe stanowią fundament dla poprawnego i niezawodnego działania całego systemu.

2.3. Podstawowe założenia systemu wspierającego pracę transportu autobusowego

W oparciu o poznane cele biznesowe powstanie dwumodułowe rozwiązanie informatyczne. Całość systemu będzie oparta na zależnych od siebie dwóch komponentach: aplikacji mobilnej wykorzystywanej przez kierowcę w czasie odbywania kursu oraz serwisu internetowego dla pasażera. Ten, używając serwisu internetowego będzie miał możliwość założenia konta i później zalogowanie się na nie. Będąc zweryfikowanym użytkownikiem serwisu uzyska dostęp do panelu pasażera, w którym przechowywane będą wszystkie aktualne i odbyte już przejazdy autobusowe. Kupując bilet na przejazd pasażer wybierać będzie interesujący go kurs z dostępnej puli przejazdów, godzinę odjazdu z przystanku startowego, jeżeli kurs jest odbywany więcej niż jeden raz na dzień, oraz z mapy. Kolejnym krokiem będzie wybór miejsc z mapy autobusu. Ostatni krok w procesie stanowi zakup i dokonanie płatności w systemie płatności internetowych. Bilety sprzedawane będą tylko poprzez serwis internetowy, nie jest przewidywana inna metoda zakupu biletów. Dane z biletów wylądują w bazie danych, z którą komunikować się będzie aplikacja mobilna dla kierowcy. Skupi ona w sobie kilka funkcji. Począwszy od aktualizowanej w czasie rzeczywistym mapy z aktualną pozycją autobusu, listą przystanków, mapą z miejscami w autobusie z informacją które są

zajęte, a które jeszcze wolne, aż do możliwości kasowania biletów poprzez zeskanowanie kodu QR wygenerowanego w poprawnie zakończonym procesie zakupu biletu.

W systemie będzie można wyróżnić czterech aktorów. Każdy z nich zdefiniował swoje cele i potrzeby:

1. Użytkownik niezalogowany:

- W przypadku, gdy będzie używał serwisu internetowego: będzie mógł się zarejestrować lub w wypadku posiadania już konta, zalogować. W obu tych przypadkach będzie możliwość „zaplanowania trasy” używając odpowiedniej opcji w serwisie.
- W przypadku, gdy będzie używał aplikacji mobilnej dla kierowcy: niezalogowany kierowca będzie mógł zgłosić problem do koordynatora tras i przejazdów.

2. Użytkownik zalogowany (pasażer):

- Ma dostęp do serwisu internetowego, gdzie może dokonywać zakupu biletów, zwracać je bez podawania przyczyny, przeglądać wszystkie dostępne trasy.

3. Użytkownik zalogowany (kierowca):

- Powinien mieć możliwość kasowania biletów poprzez skanowanie ich, by móc przyspieszyć obsługę pasażerów i organizację całego kursu.
- Możliwość podglądu na żywo wypełnienia autobusu, co pozwoli na bieżąco śledzić zapełnienie kierowanego pojazdu.
- W razie spóźnienia względem rozkładu jazdy dostaje taką informację wraz z potencjalnymi wskazówkami pomagającymi owo spóźnienie nadrobić.
- Otrzymywanie wskazówek od Koordynatora, po to, żeby móc nadrobić spóźnienie, ominąć niedogodności na trasie itp.

4. Koordynator tras i przejazdów:

- Na czas pomiędzy startem pierwszego kursu, spośród wszystkich kursów a końcem ostatniego, osoba ta potrzebuje mieć niczym nieskrępowany dostęp do bazy aktualnych kursów, by móc lepiej planować trasę oraz je dodawać i edytować w miarę potrzeb
- Śledzi aktualnie odbywających się kursów, by analizować sytuację i przekazać odpowiednie wskazówki kierowcy oraz na bieżąco interweniować w odpowiedni sposób.

Określenie, na samym początku, najważniejszych założeń i funkcjonalności, jakie ma realizować budowany system informatyczny pozwoli, osobom mającym korzystać w przyszłości z systemu, na wstępne zapoznanie się z jego użytkowymi funkcjami. Istnieje wtedy

możliwość szybkiej interwencji, w razie zaistnienia konieczności usunięcia, dodania czy też modyfikacji któregoś z aspektów budowanego systemu.

2.4. Specyfikacja wymagań funkcjonalnych systemu

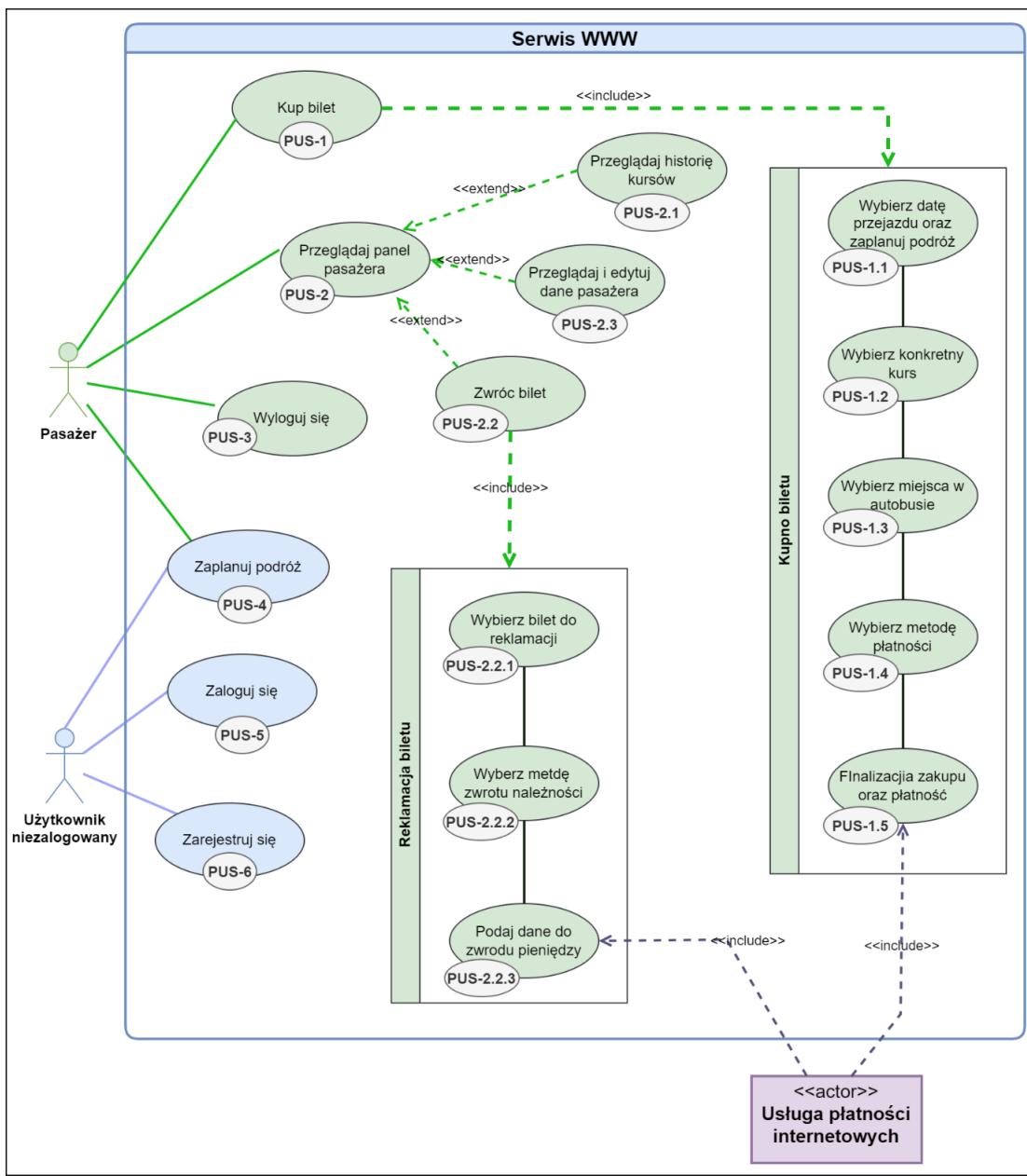
Funkcjonalność systemu wymodelowano przy użyciu jednego z najprostszych podejść, diagramów przypadków użycia. Jest to najczytelniejsza i najbardziej zrozumiała, graficzna forma prezentacji funkcjonalności budowanego systemu jaką oferuje UML. W ten sposób opisywane będą zewnętrzne sposoby zachowania się systemu z punktu widzenia użytkowników. Przypadki użycia realizowane są na rzecz aktorów, oznaczonymi na diagramach prostym, kreskowym ludzikiem.

Każdy z dwóch modułów przedstawiono osobno. Nazewnictwo przypadków użycia zostało rozpatrzone dwojako: każdy z przypadków ma swoją nazwę (zaloguj się, dokonaj płatności), oraz identyfikator. Dla serwisu internetowego powstał identyfikator w formie skrótwca „PUS” – Przypadek Użycia Serwisu, zaś dla aplikacji mobilnej „PUK” – Przypadek Użycia Kierowcy. Każdy z nich jest numerowany. Pierwszy poziom numeracji (pojedynczy numer, np. PUS-1) odnosi się do prostego przypadku użycia, wymagającego najczęściej wypełnienia pojedynczych pól czy dostania się do pojedynczej funkcji w aplikacji, np.: funkcja „skasuj bilet” w aplikacji mobilnej. Drugi poziom numeracji (PUS-2.3) rozszerza ogólne przypadki, np.: zapoczątkowuje proces reklamacji biletu. Trzeci poziom odnosi się do szczegółowych procesów, np.: przypadki PUS-2.2.1 – PUS-2.2.3 będące elementami procesu zwrotu biletu. Zaś sama opcja zmiany biletu (PUS-2.2) jest elementem rozszerzającym przypadek PUS-2 - „Przeglądaj panel pasażera”. Taka sama metodyka numerowania przypadków użycia została wykorzystana w aplikacji mobilnej dla kierowcy.

Zdefiniowane wymagania funkcjonalne budowanego systemu i oparcie ich specyfikacji na przypadkach użycia pozwoli, w późniejszych fazach rozwoju oprogramowania, na skuteczne tworzenie i wykorzystywanie scenariuszy testowych. Ich dokumentacja pozwoli testerom na dopasowanie scenariuszy testowych do wcześniej wypracowanych wymagań, będą się one pokrywały z przebiegiem zdarzeń opisanym przy każdym przypadku użycia. [12]

2.4.1. Serwis internetowy dla pasażera

W serwisie internetowym, na diagramie ujętym w niebieską ramkę, wyodrębniono dwie role w serwisie. Użytkownik niezalogowany, będzie miał dostęp tylko do funkcjonalności oznaczonej kolorem niebieskim. Użytkownik zalogowany, pasażer uzyska dostęp do pełnej funkcjonalności systemu. Na diagramie tę funkcjonalność oznaczono kolorem zielonym. Każdy z przypadków użycia jest opisany odpowiednią tabelą znajdującą się pod diagramem.



Rysunek 2-1 Diagram przypadków użycia dla serwisu internetowego (źródło: opracowanie własne)

Tabela poniżej krótko opisuje jeden z kluczowych przypadków użycia w całym systemie – „Kup bilet”, który jest procesem składającym się z kilku kroków. Podczas całego procesu

użytkownik ma możliwość wyboru połączenia, określenia preferencji dotyczących biletu, dokonania płatności i otrzymania elektronicznego potwierdzenia zakupu.

Tabela 2-1 Opis przypadku użycia "kup bilet" (źródło: opracowanie własne)

Identyfikator	PUS-1
Nazwa	Kup bilet
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5
Przebieg zdarzeń	<ol style="list-style-type: none"> Użytkownik, używając zawsze widocznego panelu znajdującego się na górnjej części serwisu internetowego wybiera opcję „zakup biletu” Wyświetlany jest odpowiedni formularz na stronie i zostaje zapoczątkowany proces kupna biletu

Pierwszy z kroków całego procesu zakupu biletu w serwisie internetowym polega na sprecyzowaniu daty, kiedy użytkownik jest zainteresowany podróżą oraz określeniu początkowego i końcowego przystanku planowanej podróży.

Tabela 2-2 Opis przypadku użycia "wybierz datę przejazdu oraz zaplanuj podróż" (źródło: opracowanie własne)

Identyfikator	PUS-1.1
Nazwa	Wybierz datę przejazdu oraz zaplanuj podróż
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, oraz PUS-1
Przebieg zdarzeń	<ol style="list-style-type: none"> W pierwszym kroku procesu zakupu biletu system wyświetla pierwszą część formularza zakupu biletu Użytkownik określa interesującą go datę odbycia kursu Użytkownik wypełnia pola „przystanek początkowy” i „przystanek końcowy” By przejść do kolejnego kroku procedury zakupu biletu, użytkownik naciska przycisk „dalej”
Alternatywny ciąg zdarzeń 1	<ol style="list-style-type: none"> System wykrywa błędne punkty trasy Miejsca z te zostają podświetlone na czerwono i pojawia się odpowiedni komunikat Użytkownik powtarza krok 3 z głównego przebiegu

Po pomyślnym określeniu szczegółów dotyczących podróży system wygeneruje listę dostępnych na dany dzień kursów. Ten przypadek użycia pozwala użytkownikowi na wybranie konkretnego kursu spośród wszystkich dostępnych opcji.

Tabela 2-3 Opis przypadku użycia "wybierz konkretny kurs" (źródło: opracowanie własne)

Identyfikator	PUS-1.2
Nazwa	Wybierz konkretny kurs
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-1, PUS-1,1
Przebieg zdarzeń	<ol style="list-style-type: none"> 1. W drugim kroku procesu zakupu biletu system wyświetla drugą część formularza zakupu biletu 2. Użytkownik w odpowiednich polach wpisuje interesujący go przystanek początkowy i końcowy. 3. Użytkownik z listy dostępnych tras wybiera ten, który go interesuje 4. Po poprawnym wyborze trasy system przechodzi do trzeciej części formularza realizującego proces zakupu biletu.
Alternatywny ciąg zdarzeń 1	<ol style="list-style-type: none"> 2a. System nie znajduje trasy przebiegającej przez podane przez użytkownika przystanki, o czym powiadamia odpowiednim komunikatem. 2b. System prosi użytkownika o podanie innych przystanków lub proponuje te, będące w najbliższej okolicy do wcześniej podanych

Mając wcześniej sprecyzowane przez użytkownika szczegóły podróży system pokaże mapę rozmieszczenia miejsc w autobusie. Miejsca wybrane, wolne i zajęte będą oznaczone odpowiednim kolorem.

Tabela 2-4 Opis przypadku użycia "wybierz miejsce / miejsca w autobusie" (źródło: opracowanie własne)

Identyfikator	PUS-1.3
Nazwa	Wybierz miejsce / miejsca w autobusie
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-1, PUS-1.1, PUS-1.2
Przebieg zdarzeń	<ol style="list-style-type: none"> 1. W trzecim kroku procesu zakupu biletu system wyświetla trzecią część formularza odpowiedzialnego za zakup biletu. 2. System wyświetla mapę miejsc autobusu. Kolorem szarym oznaczone są miejsca wolne, żółtym już zajęta, a zielonym wybrane przez pasażera. 3. Użytkownik wybiera jedno lub więcej miejsc z puli dostępnych, poprzez kliknięcie na dany numer miejsca na mapie. 4. Po poprawnym wybraniu miejsc w autobusie i zatwierdzeniu przyciskiem 'dalej' nastąpi przejście do dalszego etapu procesu zakupu biletu.
Alternatywny ciąg zdarzeń 1	<ol style="list-style-type: none"> 3a. Użytkownik nie wybrał żadnego miejsca w autobusie. 3b. System informuje o zajściu odpowiednim komunikatem nad mapą autobusu. 3c. Należy powtórzyć krok 3 z głównego przebiegu.

W ramach przypadku użycia PUS-1.4 system prezentuje użytkownikowi intuicyjną listę dostępnych metod płatności, a następnie użytkownik wybiera najdogodniejszą opcję, uwzględniając swoje preferencje, bezpieczeństwo i dostępność środków.

Tabela 2-5 Opis przypadku użycia "wybierz metodę płatności" (źródło: opracowanie własne)

Identyfikator	PUS-1.4
Nazwa	Wybierz metodę płatności
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-1, PUS-1.1, PUS-1.2, PUS-1.3
Przebieg zdarzeń	<ol style="list-style-type: none"> 1. W czwartym kroku procesu zakupu biletu system wyświetla czwartą część formularza odpowiedzialnego za zakup biletu. 2. System wyświetla użytkownikowi kilka dostępnych metod płatności. 3. Użytkownik wybiera jedną z nich. 4. System przechodzi do końcowego etapu zakupu biletu.

Przypadek użycia "Finalizacja zakupu i płatność" opisuje ostatni etap procesu zakupu biletu w serwisie internetowym. W tym etapie system prezentuje użytkownikowi podsumowanie zamówienia obejmujące: cenę za wszystkie bilety, uwzględnione zniżki i szczegóły przejazdu.

Tabela 2-6 Opis przypadku użycia "finalizacja zakupu i płatność" (źródło: opracowanie własne)

Identyfikator	PUS-1.5
Nazwa	Finalizacja zakupu i płatność
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-1, PUS-1.1, PUS-1.2, PUS-1.3, PUS-1.4
Przebieg zdarzeń	<ol style="list-style-type: none"> 1. W piątym kroku procesu zakupu biletu system wyświetla piątą część formularza zakupu biletu odpowiedzialnego za podsumowanie zamówienia i płatność. 2. System wyświetla użytkownikowi podsumowanie całego zamówienia wraz z listą metod płatności. 3. Użytkownik wybiera jedną metodę płatności. Klikając przycisk „dalej” system przechodzi do zewnętrznego serwisu dostawcy płatności.

Przypadek „PUS-2: Przeglądaj panel pasażera” opisuje interakcje zalogowanego użytkownika z jego panelem. W nim, użytkownik może przeglądać szczegóły dotyczące zakupionych biletów, odbytych kursów, czy zwracać wcześniej zakupione, ale jeszcze nie wykorzystane bilety.

Tabela 2-7 Opis przypadku użycia "przeglądaj panel pasażera" (źródło: opracowanie własne)

Identyfikator	PUS-2
Nazwa	Przeglądaj panel pasażera
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5

Przebieg zdarzeń	1. Użytkownik z górnego menu w postaci zawsze widocznej belki wybiera opcję „panel klienta”. 2. System wyświetla panel klienta z kilkoma możliwymi opcjami do wyboru.
-------------------------	--

Uzyskawszy dostęp do panelu pasażera, użytkownik ma dostęp do historii i szczegółów swoich wcześniejszych przejazdów autobusowych. Może je przeglądać, wyświetlać szczegóły i ponownie kupić bilet na wcześniej odbytą trasę, pod warunkiem, że ta nadal będzie dostępna w systemie.

Tabela 2-8 Opis przypadku użycia "przeglądaj historię kursów" (źródło: opracowanie własne)

Identyfikator	PUS-2.1
Nazwa	Przeglądaj historię kursów (przypadek rozszerzający PUS-2)
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-2
Przebieg zdarzeń	1. Użytkownik, wcześniej wchodząc w „panel klienta” wybiera opcję „Przeglądaj historię kursów” 2. System wyświetla listę ze wszystkimi dotychczasowymi kursami
Alternatywny ciąg zdarzeń 1	2a. System stwierdza brak kursów do wyświetlenia, o czym informuje stosownym komunikatem.

Użytkownik mając kupiony, ale jeszcze nieużyty bilet może go zwrócić. Opcja „zwróć bilet” zapoczątkowuje całą, kilkuetapową procedurę zwracania należności za bilet. Wydzielenie funkcjonalności w panelu użytkownika pozwoli na efektywne zgłoszenie reklamacji, unikając zbędnych trudności i ułatwiając proces obsługi klienta.

Tabela 2-9 Opis przypadku użycia "zwróć bilet" (źródło: opracowanie własne)

Identyfikator	PUS-2.2 (przypadek rozszerzający PUS-2)
Nazwa	Zwróć bilet
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-2, PUS-1
Przebieg zdarzeń	1.Użytkownik, wcześniej wchodząc w „panel klienta” wybiera opcję „zwróć bilet”. 2.System wyświetla listę dostępnych i nieskasowanych biletów, które można będzie zwrócić. 3.System rozpoczyna proces zwrotu biletu
Alternatywny ciąg zdarzeń 1	2a – System stwierdza brak biletów do wyświetlenia, o czym informuje stosownym komunikatem w miejscu, w którym powinna wyświetlić się lista biletów pasażera.

Na samym początku procedury zwrotu biletu użytkownik, z listy dostępnych i możliwych do zwrotu, wybiera ten, który go interesuje. Po podjęciu decyzji przez użytkownika, system przejdzie do kolejnego kroku.

Tabela 2-10 Opis przypadku użycia "wybierz bilet do reklamacji" (źródło: opracowanie własne)

Identyfikator	PUS-2.2.1
Nazwa	Wybierz bilet do reklamacji (przypadek zawarty w PUS-2.2)
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-2, PUS-1, PUS-2.2
Przebieg zdarzeń	<ol style="list-style-type: none"> 1. W pierwszym kroku procesu zwrotu biletu użytkownik, wcześniej wybierając opcję „zwróć bilet” wybiera bilet podlegający zwrotowi. 2. System wyświetla szczegóły wybranego biletu. 3. Użytkownik klikając przycisk „zwróć bilet” przechodzi do kolejnego kroku procesu.

Po wybraniu biletu użytkownik jest proszony o wybór preferowanej metody zwrotu należności w związku z dokonaną wcześniej płatnością za bilet. Wyświetlane jest kilka opcji zwrotu zapłaty do wyboru.

Tabela 2-11 Opis przypadku użycia "wybierz metodę zwrotu należności" (źródło: opracowanie własne)

Identyfikator	PUS-2.2.2
Nazwa	Wybierz metodę zwrotu należności (przypadek zawarty w PUS-2.2)
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-2, PUS-1, PUS-2.2, PUS-2.2.1
Przebieg zdarzeń	<ol style="list-style-type: none"> 1. W drugim kroku procesu, w pojawiającej się drugiej części formularza użytkownik wybiera jedną z dostępnych form zwrotu należności za bilet. 2. Po wybraniu jednej z nich system przechodzi do trzeciego kroku procesu.

Mając sprecyzowaną metodę zwrotu zapłaty, użytkownik podaje dane, konieczne do zwrotu, odpowiednie dla wcześniej wybranej opcji. Pieniądze na niewykorzystany i zwrócony bilet trafiają na konto pasażera w terminie określonym przez dostawcę usługi płatności.

Tabela 2-12 Opis przypadku użycia "podaj dane do zwrotu pieniędzy" (źródło: opracowanie własne)

Identyfikator	PUS-2.2.3
Nazwa	Podaj dane do zwrotu pieniędzy (przypadek zawarty w PUS-2.2)
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-2, PUS-1, PUS-2.2, PUS-2.2.1, PUS-2.2.2,

Przebieg zdarzeń	1.W trzecim kroku procesu i trzeciej części formularza zwrotu należności za bilet użytkownik podaje dane do zwrotu odpowiednie dla formy, którą wybrał w poprzednim kroku. 2.System komunikuje się z dostawcą usług płatności i następuje zwrot środków jedną z wcześniej wybranych metod.
Alternatywny ciąg zdarzeń 1	1a. System stwierdza błąd w formularzu zwrotu należności za bilet. 1b. Pola z niepoprawnymi danymi zostają podświetlone kolorem czerwonym 1c. Użytkownik powtarza krok 1 głównego przebiegu.

Ostatnią funkcjonalnością dostarczaną przez panel pasażera będzie możliwość przejrzenia i edycji danych zalogowanego pasażera, oraz zmiana hasła.

Tabela 2-13 Opis przypadku użycia "przeglądaj i edytuj dane pasażera" (źródło: opracowanie własne)

Identyfikator	PUS-2.3 (przypadek rozszerzający PUS-2)
Nazwa	Przeglądaj i edytuj dane pasażera
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5, PUS-2
Przebieg zdarzeń	1.System wyświetla dane aktualnie zalogowanego pasażera 2.Użytkownik naciska przycisk „edytuj”. 3.System wyświetla formularz edycji danych użytkownika. 4.Użytkownik edytuje pole, które go interesują. 5.System weryfikuje i zapisuje wprowadzone dane
Alternatywny ciąg zdarzeń 1	4a – System stwierdza niezgodność danych prowadzonych przez użytkownika. Pole wymagające korekty zostaje podświetlone na czerwono. 4b – Użytkownik powtarza krok 4 głównego przebiegu.

Użytkownik skończywszy wszystkie działanie w serwisie internetowym może się wylogować. Przypadek użycia "Wyloguj się" umożliwia użytkownikowi zakończenie sesji i przejście do strony głównej serwisu.

Tabela 2-14 Opis przypadku użycia "wyloguj się" (źródło: opracowanie własne)

Identyfikator	PUS-3
Nazwa	Wyloguj się
Aktorzy	Użytkownik zalogowany
Warunki wstępne	Użytkownik wykonał PUS-5
Przebieg zdarzeń	1.Użytkownik naciska przycisk „Wyloguj się” znajdujący się w prawej górnej stronie serwisu 2.System wyloguje użytkownika i przechodzi do ekranu głównego serwisu

Opcja Zaplanuj podróż jest dostępna zarówno dla osób zalogowanych, jak i niezalogowanych. Umożliwia użytkownikowi zaplanowanie swojej podróży autobusem

poprzez wybór odpowiednich parametrów, takich jak przystanek początkowy i docelowy oraz data wyjazdu.

Tabela 2-15 Opis przypadku użycia "zaplanuj podróż" (źródło: opracowanie własne)

Identyfikator	PUS-4
Nazwa	Zaplanuj podróż
Aktorzy	Użytkownik zalogowany, użytkownik niezalogowany
Przebieg zdarzeń	1.Użytkownik wybiera opcję „zaplanuj podróż” z menu zlokalizowanego w górnej części serwisu 2.System przechodzi do formularza odpowiedzialnego za określenie punktów podróży. 3.Użytkownik wypełnia pola „przystanek początkowy” oraz „przystanek końcowy”. 4.System wyświetla listę tras spełniających zadane kryterium wyszukiwania.
Alternatywny ciąg zdarzeń 1	4a. System nie znajduje zgodnej trasy z podanymi przez użytkownika kryteriami, wyświetla się odpowiedni komunikat. 4b. Użytkownik jest proszony o powtórzenie czynności zapoczątkowanej w punkcie 3 głównego przebiegu.

By użytkownik mógł w pełni korzystać ze wszystkich funkcji serwisu, musi zalogować się na wcześniej utworzone przez siebie konto. Po zalogowaniu się system załaduje wszystkie dane użytkownika, takie jak: historię zakupów biletów, dane osobowe i inne.

Tabela 2-16 Opis przypadku użycia "zaloguj się" (źródło: opracowanie własne)

Identyfikator	PUS-5
Nazwa	Zaloguj się
Aktorzy	Użytkownik niezalogowany
Warunki wstępne	Użytkownik wykonał PUS-6
Przebieg zdarzeń	1.Użytkownik wybiera opcję „Zaloguj / zarejestruj się” znajdująca się w prawej górnej części serwisu. 2.System przechodzi do okna logowania. 3.Użytkownik wypełnia pola: „email” oraz „hasło”. 4.Po pomyślnym przejściu weryfikacji, system zezwala na pełne korzystanie z serwisu.
Alternatywny ciąg zdarzeń 1	4a. System nie jest w stanie zweryfikować użytkownika, o czym informuje odpowiednim komunikatem. 4b. Użytkownik jest proszony o powtórzenie kroku 3 głównego przebiegu.

Przypadek użycia PUS-5 skupia się na opisie procesu rejestracji nowego użytkownika serwisu. Stworzenie indywidualnego konta pozwoli użytkownikowi na przeglądanie historii

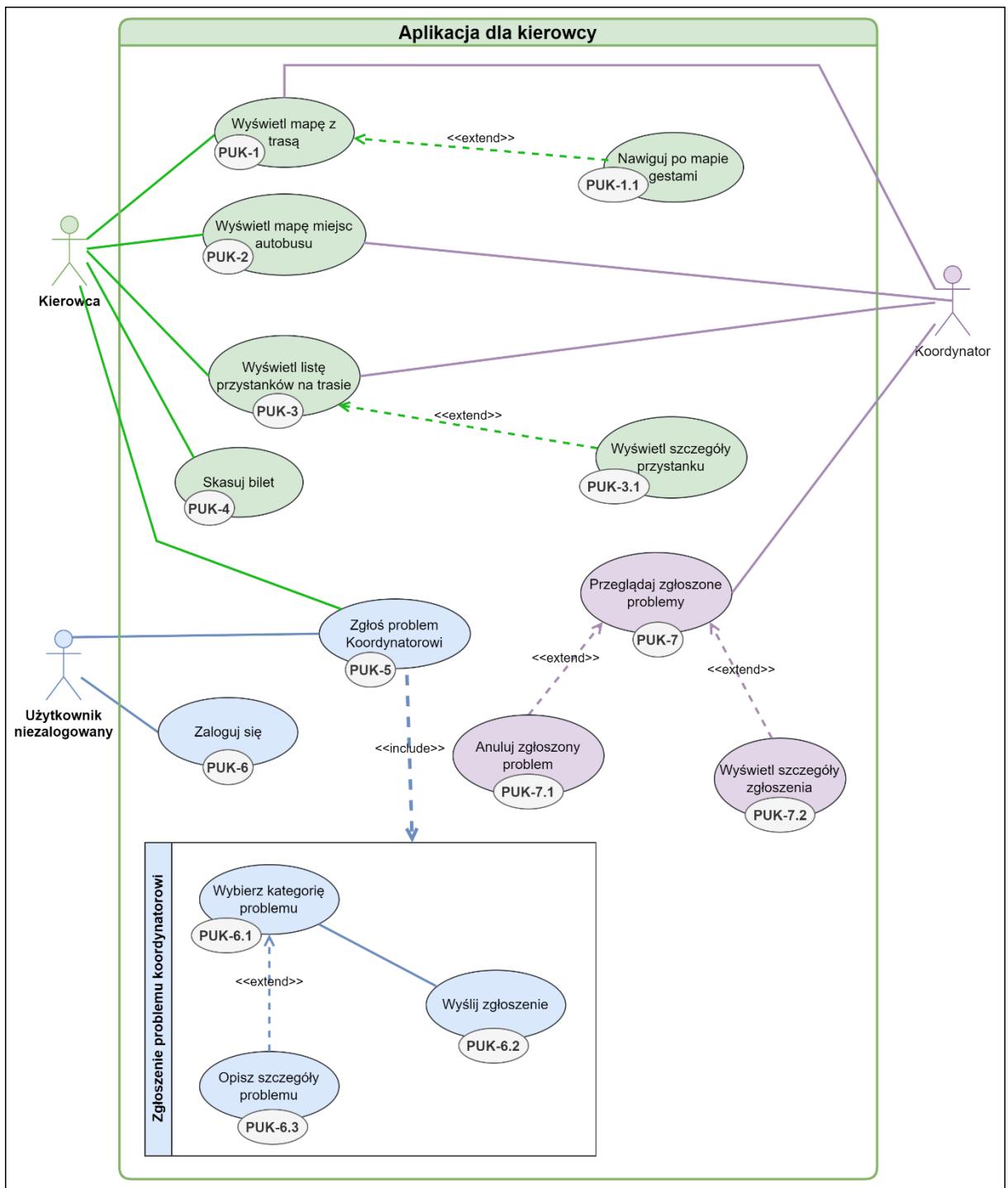
przejazdów, zarządzanie swoimi danymi osobowymi, a także otrzymywanie informacji o promocjach czy ofertach specjalnych.

Tabela 2-17 Opis przypadku użycia "zarejestruj się" (źródło: opracowanie własne)

Identyfikator	PUS-5
Nazwa	Zarejestruj się
Aktorzy	Użytkownik niezalogowany
Warunki wstępne	Brak
Przebieg zdarzeń	1.Użytkownik wybiera opcję „Zaloguj / zarejestruj się” znajdująca się w prawej górnej części serwisu. 2.System wyświetla formularz rejestracji. 3.Użytkownik wypełnia formularz rejestracyjny. 4.System wysyła wiadomość e-mail w celu weryfikacji rejestrowanego użytkownika.

2.4.2. Aplikacja mobilna dla kierowcy

Analogicznie do podejścia w poprzednim podrozdziale, funkcjonalność aplikacji mobilnej dla kierowcy przedstawia diagram przypadków użycia. W drugiej części systemu zostały wyodrębnione trzy role w systemie. Użytkownik niezalogowany, na diagramie oznaczony kolorem niebieskim, użytkownik zalogowany, kierowca, kolor zielony oraz Koordynator z oznaczeniem jego przypadków użycia kolorem fioletowym. Każdy z przypadków użycia ujętych w diagramie ma swoją osobną tabelę opisującą, zlokalizowaną pod diagramem.



Rysunek 2-2 Diagram przypadków użycia dla aplikacji mobilnej (źródło: opracowanie własne)

Jedną z funkcjonalności aplikacji mobilnej dla kierowcy jest możliwość wyświetlenia na mapie pełnej trasy, którą ma pokonać podczas swojego kursu. Dla kierowcy jest jedna z kluczowych funkcji podczas pracy, umożliwiająca zapoznanie się z drogą i jej przebiegiem w czasie rzeczywistym, oraz reagowanie na nieprzewidziane zdarzenia.

Tabela 2-18 Opis przypadku użycia "wyświetl mapę z trasą" (źródło: opracowanie własne)

Identyfikator	PUK-1
Nazwa	Wyświetl mapę z trasą
Aktorzy	Kierowca, koordynator
Warunki wstępne	Użytkownik wykonał PUK-5
Przebieg zdarzeń	1.Użytkownik, używając panelu górnego, wchodzi w opcję „Widok trasy” 2.System pobiera dane lokalizacyjne oraz dane kursu. 3.Wyświetla się okno z mapą aktualnego kursu danego kierowcy i aktualną pozycją pojazdu odświeżaną co sekundę
Przypadki rozszerzające	PUK-1.1 Nawiguj po mapie gestami 1. Gestem uszczypnięcia mapa zostanie oddalona, gestem rozszerzenia palców, przybliżona
Alternatywny ciąg zdarzeń 1	2a. System wykrywa brak połączenia z siecią 3a. W wyniku braku połączenia z Internetem mapa wyświetla się w trybie ograniczonym – pojawia się czerwona ramka dookoła okna z mapą. 4a. System samodzielnie będzie próbował odnowić połączenie z siecią. Po nawiązaniu połączenia nastąpi powrót do punktu 2 głównego przebiegu.
Alternatywny ciąg zdarzeń 2	2a. System wykrywa brak połączenia z systemami lokalizacji 2b. W wyniku braku połączenia z systemami lokalizacji położenie pojazdu nie jest dokładne. Informacja o takim stanie jest wyświetlana w wyskakującym okienku tuż pod belką z przyciskami nawigacyjnymi aplikacji, ponadto po prawej stronie okna widnieć będzie przekreślony symbol nawigacji GPS na czas braku dostępności do niej. 2c. System samodzielnie będzie próbował odnowić połączenie z systemami lokalizacji. Po nawiązaniu połączenia nastąpi powrót do punktu 2 głównego przebiegu.

Kierowca mając do dyspozycji funkcjonalność odpowiedzialną za monitorowanie zajętości siedzeń pojazdu przez pasażerów może kontrolować na bieżąco stopień jego zapełnienia. System będzie, podczas postoju na przystankach, zmieniał statusy siedzeń po zeskanowaniu biletu, na zajete, bądź wolne. Pasażer wychodzący na swoim docelowym przystanku zwalnia miejsce w pojeździe, co kierowca zobaczy na mapie. Takie rozwiązanie pozwoli też uniknąć pasażerów jadących „na gapę”.

Tabela 2-19 Opis przypadku użycia "wyświetl mapę miejsc autobusu" (źródło: opracowanie własne)

Identyfikator	PUK-2
Nazwa	Wyświetl mapę miejsc autobusu
Aktorzy	Kierowca, koordynator
Warunki wstępne	Użytkownik wykonał PUK-5
Przebieg zdarzeń	1.Użytkownik, używając panelu górnego, wchodzi w opcję „Mapa miejsc”

	2.Pojawia się widok z odwzorowaniem miejsc w pojeździe: kolorem szarym oznaczone są miejsca wolne, kolorem niebieskim – zajęte przez pasażera
Alternatywny ciąg zdarzeń 1	<p>2a. System wykrywa brak połączenia z siecią</p> <p>3a. W wyniku braku połączenia z Internetem mapa wyświetla się w trybie ograniczonym – pojawia się czerwona ramka dookoła okna z mapą.</p> <p>4a. System samodzielnie będzie próbował odnowić połączenie z siecią. Po nawiązaniu połączenia nastąpi powrót do punktu 2 głównego przebiegu.</p>

Kierowca, z poziomu aplikacji, będzie miał dostęp do listy przystanków aktualnie odbywanego kursu. Na liście wyświetlane będą godziny przyjazdu i odjazdu wraz z potencjalnymi różnicami czasowymi.

Tabela 2-20 Opis przypadku użycia "wyświetl listę przystanków na trasie" (źródło: opracowanie własne)

Identyfikator	PUK-3
Nazwa	Wyświetl listę przystanków na trasie
Aktorzy	Kierowca, koordynator
Warunki wstępne	Użytkownik wykonał PUK-5
Przebieg zdarzeń	<p>1.Użytkownik, używając panelu górnego, wchodzi w opcję „Lista przystanków”</p> <p>2.Pojawia się widok z listą wszystkich przystanków na danej trasie. Pojedynczy wiersz składa się odpowiednio z: nazwy przystanku, godziny przyjazdu, godziny odjazdu</p>
Przypadki rozszerzające	<p>PUK-3.1 Wyświetl szczegóły przystanku</p> <p>1.Użytkownik z listy przystanków wybiera dotykiem jeden z nich</p> <p>2.Pojawia się okno z dodatkowymi informacjami na temat wybranego przystanku: jego lokalizacji (ulica), ilości pasażerów wsiadających i wysiadających</p> <p>3.Aby zamknąć okno z dodatkowymi informacjami należy dotknąć miejsce znajdujące się poza jego obrębem</p>
Alternatywny ciąg zdarzeń 1	<p>2a. System wykrywa brak połączenia z siecią</p> <p>2b. W wyniku braku połączenia z Internetem mapa wyświetla się w trybie ograniczonym – pojawia się czerwona ramka dookoła okna z mapą.</p> <p>2c. System samodzielnie będzie próbował odnowić połączenie z siecią. Po nawiązaniu połączenia nastąpi powrót do punktu 2 głównego przebiegu.</p>

Funkcja kasowania biletu będąca integralną częścią całego systemu powoli kierowcy na sprawną obsługę pasażerów. Przyczyni się to do płynnej i sprawnej organizacji podróży, co będzie miało wpływ na zadowolenie i komfort pasażerów korzystających z usług transportowych.

Tabela 2-21 Opis przypadku użycia "skasuj bilet" (źródło: opracowanie własne)

Identyfikator	PUK-4
Nazwa	Skasuj bilet
Aktorzy	Kierowca
Warunki wstępne	Użytkownik wykonał PUK-5
Przebieg zdarzeń	1.Użytkownik z górnego menu wybiera opcję „Skasuj bilet” 2.Uruchamia się aparat urządzenia 3.Użytkownik skieruje urządzenie na kod QR znajdujący się w prawej części biletu 4.Bilet został zeskanowany. Urządzenie sygnalizuje zeskanowanie biletu odpowiednim komunikatem. 5.Aby zmienić wyświetlane okno użytkownik wybiera inną opcję z górnego menu
Alternatywny ciąg zdarzeń 1	3a. System nie rozpoznaje kodu QR 3b. Użytkownik zostaje o tym poinformowany odpowiednim komunikatem 3c. Należy powtórzyć procedury zapoczątkowane w punkcie 3 przebiegu głównego zdarzeń

Kierowca, który wcześniej dopełnił wymaganych formalności z firmą transportową może się zalogować do aplikacji nadanym mu numerem oraz wybranym przez siebie hasłem. Dopiero po zweryfikowaniu kierowcy system umożliwi w pełni korzystanie ze wszystkich funkcji aplikacji dla kierowcy.

Tabela 2-22 Opis przypadku użycia "zaloguj się" (źródło: opracowanie własne)

Identyfikator	PUK-5
Nazwa	Zaloguj się
Aktorzy	Użytkownik niezalogowany
Warunki wstępne	Użytkownik został dodany do bazy kierowców
Przebieg zdarzeń	1.Włączając aplikację na urządzeniu użytkownikowi pojawia się okno z dwoma komórkami do wypełnienia: numer kierowcy oraz hasło 2.Użytkownik wypełnia je danymi i naciska przycisk „zaloguj” 3.Użytkownik został zalogowany do systemu i pojawia mu się główne okno aplikacji
Alternatywny ciąg zdarzeń 1	2a. Użytkownik podał niepoprawne dane 2b. Użytkownik zostaje poinformowany odpowiednim komunikatem oraz zostają podświetlone komórki z wprowadzonymi niepoprawnymi danymi 2c. Użytkownik zostaje poproszony o poprawę danych wprowadzonych w kroku 2 z PUK-5
Alternatywny ciąg zdarzeń 2	2a. Brak możliwości zalogowania wynikający z braku połączenia z siecią 2b. Użytkownik włączając aplikację zostaje poinformowany o braku połączenia z siecią odpowiednim komunikatem

	<p>2c. Użytkownik powinien się skontaktować z Koordynatorem tras i przejazdów w celu uzyskania loginu i hasła umożliwiającego tymczasowe korzystanie z aplikacji w trybie offline</p> <p>2d. Po odzyskaniu połączenia z siecią aplikacja ponownie zapyta o dane do logowania. Nastąpi powrót do punktu 1 w PUK-5</p>
--	--

Będąc zarówno zalogowanym, jak i nie, kierowca będzie mógł zgłosić, poprzez formularz w aplikacji, problem osobie odpowiedzialnej za koordynowanie przejazdów i reagowanie na incydenty podczas nich.

Tabela 2-23 Opis przypadku użycia "zgłoś problem koordynatorowi" (źródło: opracowanie własne)

Identyfikator	PUK-6
Nazwa	Zgłoś problem koordynatorowi
Aktorzy	Użytkownik niezalogowany, kierowca
Warunki wstępne	brak
Przebieg zdarzeń	<p>1.Użytkownik wybiera opcję „Zgłoś problem koordynatorowi” dostępną w dolnej części ekranu logowania lub po zalogowaniu – w prawym górnym roku ekranu aplikacji.</p> <p>2.System przechodzi do formularza zgłaszania problemu.</p>

Na początku zgłaszania problemu, kierowca zostanie poproszony o jego sprecyzowanie. Z listy z kategoriami zgłoszeń wybierze odpowiednią dla danej sytuacji. Takie zgłoszenie, jeżeli nie wymaga bardziej szczegółowego opisu, będzie można wysłać do osoby odpowiedzialnej za ich rozpatrywanie.

Tabela 2-24 Opis przypadku użycia "wybierz kategorię problemu" (źródło: opracowanie własne)

Identyfikator	PUK-6.1
Nazwa	Wybierz kategorię problemu
Aktorzy	Użytkownik niezalogowany, kierowca
Warunki wstępne	brak
Przebieg zdarzeń	<p>1.Z rozwijanej listy „kategoria problemu” użytkownika wybiera tę dotyczącą lub najbardziej zbliżoną do sytuacji w jakiej się znalazł.</p> <p>2.Zgłoszenie jest gotowe do zapisania i poinformowania o nim koordynatora tras i przejazdów.</p>

Gotowe zgłoszenie, które nie wymaga bardziej szczegółowego opisu, może zostać wysłane w celu rozpatrzenia i podjęcia ewentualnej interwencji przez osobę przeznaczoną do tego typu zadań w przedsiębiorstwie zajmującym się przewozem autobusowym osób.

Tabela 2-25 Opis przypadku użycia "wyślij zgłoszenie" (źródło: opracowanie własne)

Identyfikator	PUK-6.2
Nazwa	Wyślij zgłoszenie
Aktorzy	Użytkownik niezalogowany, kierowca
Warunki wstępne	Użytkownik wykonał PUK-6
Przebieg zdarzeń	1.Użytkownik mając skompletowane zgłoszenie naciska przycisk „wyślij zgłoszenie” 2.System zapisuje zgłoszenie w bazie danych i powiadamia o tym koordynatora tras i przejazdów

W przypadku, gdy kierowca uzna, że dostępne kategorie będą niewystarczające do ukazania wyjątkowej sytuacji, w której się znalazł, może, w osobnym oknie bardziej szczegółowo ją opisać.

Tabela 2-26 Opis przypadku użycia "opisz szczegóły problemu" (źródło: opracowanie własne)

Identyfikator	PUK-6.3
Nazwa	Opisz szczegóły problemu
Aktorzy	Użytkownik niezalogowany, kierowca
Warunki wstępne	Użytkownik wykonał PUK-6
Przebieg zdarzeń	1.W formularzu zgłaszenia problemu użytkownik wchodzi w pole tekstowe „Szczegóły problemu” 2.W polu tekstowym, jeżeli wymaga tego sytuacja, a kategorie problemu z rozwijanej listy są niewystarczające, opisuje szczegóły, czego dotyczy zgłoszenie.

Wszystkie zgłoszenia docierają do „koordynatora tras i przejazdów”. Z poziomu aplikacji przeglądowej osoba ta będzie miała dostęp do widoku listy ze wszystkimi zgłoszonymi sytuacjami przez kierowców.

Tabela 2-27 Opis przypadku użycia "przeglądaj zgłoszone problemy" (źródło: opracowanie własne)

Identyfikator	PUK-7
Nazwa	Przeglądaj zgłoszone problemy
Aktorzy	Koordynator tras i przejazdów
Warunki wstępne	Lista zgłoszonych problemów nie jest pusta
Przebieg zdarzeń	1.Użytkownik wchodzi w opcję „Zgłoszone problemy” 2.System wyświetla listę wszystkich zgłoszonych, nierożpatrzonych problemów. 3.Użytkownik wybiera ten, którego chce wyświetlić szczegóły 4.Szczegóły zdarzenia będą widoczne w oknie po prawej stronie listy ze zgłoszeniami.

Alternatywny ciąg zdarzeń 1	2a. System stwierdza brak problemów do wyświetlania. 2b. W miejscu, w którym powinny znaleźć się dane ze zgłoszeniami widnieć będzie komunikat odpowiedni do sytuacji.
------------------------------------	---

Zgłoszony przez kierowcę incydent, który nie wymaga dalszego procedowania, może zostać anulowany przez koordynatora tras i przejazdów z powodu rozwiązania problemu.

Tabela 2-28 Opis przypadku użycia "anuluj zgłoszony problem" (źródło: opracowanie własne)

Identyfikator	PUK-7.1
Nazwa	Anuluj zgłoszony problem
Aktorzy	Koordynator tras i przejazdów
Warunki wstępne	Lista zgłoszonych problemów nie jest pusta, użytkownik wykonał PUK-7
Przebieg zdarzeń	1.Użytkownik wybiera interesujące go zdarzenie. 2.Zdarzenie po wcześniejszym rozpatrzeniu, może anulować naciskając przycisk „anuluj zdarzenie”

W przypadku, gdy żadna z osób odpowiadających za reagowanie na incydenty w przedsiębiorstwie transportu autobusowego nie będzie w stanie rozwiązać problemu, takowy można przekierować dalej, na przykład do odpowiednich służb. Taką sytuację powinno się odnotować w systemie.

Tabela 2-29 Opis przypadku użycia "wyślij zgłoszenie problemu dalej" (źródło: opracowanie własne)

Identyfikator	PUK-7.2
Nazwa	Wyślij zgłoszenie problemu dalej
Aktorzy	Koordynator tras i przejazdów
Warunki wstępne	Lista zgłoszonych problemów nie jest pusta, użytkownik wykonał PUK-7
Przebieg zdarzeń	1.Użytkownik wybiera interesujące go zdarzenie. 2.Użytkownik zapoznaje się z treścią zdarzenia. 3.Po przeanalizowaniu treści, użytkownik może przesyłać treść zdarzenia dalej, np. do odpowiednich służb.

2.5. Specyfikacja wymagań pozafunkcjonalnych systemu

Wymagania pozafunkcjonalne obejmują zakres aspektów nie dotyczących bezpośrednio elementów funkcjonalnych systemu, ale wpływają na niego często w bardzo znaczny sposób, jak na przykład określenie przedziałów czasowych, kiedy system może nie funkcjonować, by ten poddać konserwacji czy aktualizacjom. Mają znaczenie dla wydajności i niezawodności budowanego systemu, oraz jego ogólnej jakości i doświadczeń płynących z użytkowania i pracy z nim.

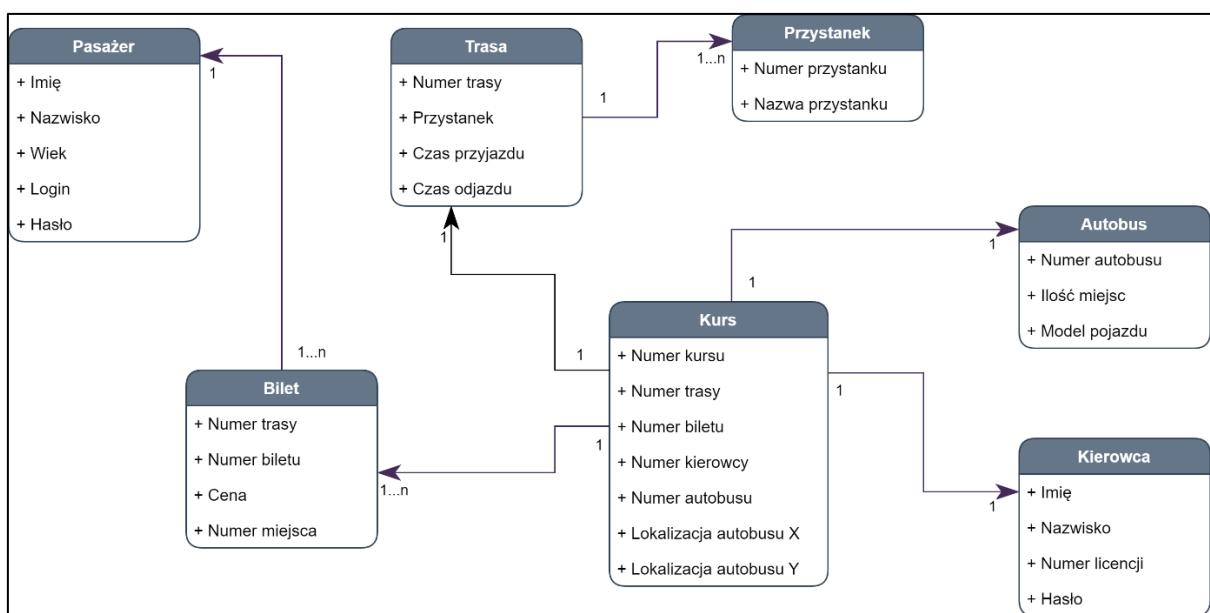
1. Interfejs użytkownika w serwisie przeglądarkowym powinien być czytelny dla osób z zaburzeniami widzenia – paleta kolorów zostanie dostosowana do osób z najczęściej występującymi zaburzeniami postrzegania kolorów – deuteranopii polegającej na mylnym rozpoznawaniu barwy zielonej lub mylenia jej z czerwoną.
2. Kolorystyka i rozmiar interfejsu użytkownika w aplikacji mobilnej dla kierowcy powinny być zaprojektowane tak, by mogły być czytelne w pełnym słońcu oraz dostosowane do wygodniej obsługi w trakcie trwania przejazdu autobusu.
2. Działanie systemu będzie zgodne z aktualnie obowiązującym prawem:
 - a. Ustawa z dnia 16 grudnia 2010 r. o publicznym transporcie zbiorowym. Określa ona ramy i ogólne zasady funkcjonowania regularnego przewozu osób transportem zbiorowym na terenie kraju.
 - b. Ustawa z dnia 20 czerwca 1992 r. o uprawnieniach do ulgowych przejazdów środkami publicznego transportu zbiorowego
 - c. Ustawa z dnia 15 listopada 1984 r. Prawo przewozowe. Akt określa za przewóz osób oraz inne opłaty za usługę przewozu osób przez operatora.
 - d. Ustawa z dnia 10 maja 2018 r. o ochronie danych osobowych
 - e. Ustawa z dnia 16 lipca 2004 r. Prawo telekomunikacyjne, szczególnie art. 173 i 174 (cookies)
3. Przewiduje się, że z systemu będzie korzystało:
 - a. Pomiędzy 17-25 kierowców będących aktualnie w trasie
 - b. Czterech koordynatorów
 - c. Jeden administrator
 - d. Powyższe wartości w przyszłości będą zmienne - zależnie od rozbudowy Przedsiębiorstwa jak i samego systemu
4. System musi być dostępny od wczesnych godzin porannych (4:30) do późnego wieczora (obecnie okolice północy):
 - a. Godziny te mogą się zmieniać w trakcie angażowania nowych kierowców i zmiany lub dodawania nowych tras
 - b. Całość wymiany danych odbywa się poprzez sieć
5. Konserwacja i wdrażanie nowych funkcjonalności zaplanowana zostanie w godzinach, kiedy system jest najmniej używany, tj. w godzinach niekursowania autobusów
6. Wymagania sprzętowe aplikacji dla kierowcy
 - a. W celu zapewnienia ciągłego dostępu do sieci wymaganego do poprawnego działania aplikacji należy wyposażyć kierowców w urządzenia obsługujące

różne możliwości połączenia się z siecią (3G, 4G) posiadające system Google Android w wersji 9 i wyższej.

Wyspecyfikowanie wymagań pozafunkcjonalnych koncentruje się na kluczowych aspektach systemu, które nie dotyczą bezpośrednio jego funkcji, ale mają istotny wpływ na jego jakość i użyteczność.

2.6. Modelowanie danych przy użyciu diagramu encji

Na podstawie wcześniej wyspecyfikowanych wymagań funkcjonalnych i pozafunkcjonalnych powstał wstępny model danych, z jakich będzie korzystał system. Centralną jego częścią będzie tabela „Kurs” agregująca wszystkie niezbędne informacje z innych tabel, takich jak „Trasa”, „Autobus”, czy „Kierowca”.



Rysunek 2-3 Diagram encji (źródło: opracowanie własne)

Tabela „pasażer” w przyszłej bazie danych reprezentowana przez encję przechowuje będzie wymagane dane pasażera, które będą konieczne przy zalogowaniu się do serwisu, korzystania w całości z niego i kupna biletu.

Tabela 2-30 Opis encji "pasażer" (źródło: opracowanie własne)

Nazwa encji	Pasażer
Nazwa i opis atrybutów	Imię - jedna z wymaganych danych pasażera, będzie umieszczona na bilecie Nazwisko - jedna z wymaganych danych pasażera, będzie umieszczona na bilecie Wiek - jedna z wymaganych danych pasażera, będzie umieszczona na bilecie, na jego podstawie wyliczana i przydzielana będzie zniżka na przejazd

	Email - wymagany do weryfikacji konta i wysyłki elektronicznego biletu i logowania się na konto
	Hasło - wymaganie pole do logowania się do serwisu, zgodne z polityką haseł

W przypadku zdecydowania się na przejazd, pasażer kupuje bilet. Wypełnia formularz niezbędnymi danymi. Ilość biletów nie może przekraczać ilości miejsc w autobusie wyznaczonym do odbycia kursu, którego dotyczy dany bilet.

Tabela 2-31 Opis encji "bilet" (źródło: opracowanie własne)

Nazwa encji	Bilet
Nazwa i opis atrybutów	Numer trasy - każdy bilet jest przeznaczony na daną trasę, będącą wcześniej wprowadzoną i do bazy
	Numer biletu - każdy bilet ma swój indywidualny identyfikator w formacie: numer trasy - przystanek startowy- godzina zakupu
	Cena biletu - wyliczana na podstawie ceny bazowej i wieku pasażera
	Numer miejsca - wybrany przez pasażera w formularzu w serwisie internetowym

Wszystkie aktualne przystanki, na których zatrzymują się autobusy przewoźnika. Każdy z nich ma swój identyfikator i nazwę wziętą od nazwy ulicy, lub bezpośredniego środowiska, gdzie się znajduje.

Tabela 2-32 Opis encji "przystanek" (źródło: opracowanie własne)

Nazwa encji	Przystanek
Nazwa i opis atrybutów	Numer przystanku - każdemu przystankowi wprowadzonemu do bazy nadaje się niepowtarzalny jego numer
	Nazwa przystanku - nazwa wprowadzona do bazy powstaje na podstawie nazwy ulicy, na którym znajduje się przystanek, bądź popularnego punktu orientacyjnego będącego w pobliżu lokalizacji przystanku

Na bieżąco aktualizowana flota pojazdów. Każdy z nich ma swój identyfikator, ilość miejsc (która nie może zostać przekroczena), oraz model. Jest przypisywany do kursu.

Tabela 2-33 Opis encji "autobus" (źródło: opracowanie własne)

Nazwa encji	Autobus
Nazwa i opis atrybutów	Numer autobusu - każdy autobus ma swój unikatowy identyfikator
	Ilość miejsc - jest uzależniona od modelu pojazdu, i uzależnia ilość pasażerów mogących przejechać danym kursem

	Model pojazdu - marka pojazdu, jedynie do celów informacyjnych dla pasażera
--	--

By móc funkcjonować w firmie musi legitymować się ważną licencją i posiadać aktualne prawo jazdy kategorii D. jest przypisywany do danego autobusu, który w danej chwili ma przypisany jakiś kurs. Do aplikacji loguje się numerem licencji oraz ustanowionym przez siebie hasłem

Tabela 2-34 Opis encji "Kierowca" (źródło: opracowanie własne)

Nazwa encji	Kierowca
Nazwa i opis atrybutów	Imię - pole wymagane jedynie do celów informacyjnych kierownictwa firmy
	Nazwisko - pole wymagane jedynie do celów informacyjnych kierownictwa firmy
	Numer licencji - wymagany do logowania do aplikacji mobilnej
	Hasło - wymagane do logowania do aplikacji mobilnej

Trasa ma swój identyfikator. Listę przystanków, czas przyjazdu na przystanek i odjazdu z niego. By autobus mógł kursować na trasie, musi mieć ona co najmniej 5 przystanków.

Tabela 2-35 Opis encji "Trasa" (źródło: opracowanie własne)

Nazwa encji	Trasa
Nazwa i opis atrybutów	Numer trasy - każda wprowadzona do bazy i zatwierdzona trasa ma swój identyfikator
	Przystanek - każda trasa ma określony i zatwierdzony ciąg następujących po sobie przystanków
	Czas przyjazdu - czas odjazdu z danego miejsca na trasie
	Czas odjazdu - czas przyjazdu do danego miejsca na trasie

Każdy kurs ma swój identyfikator. Odbędzie się wtedy, gdy: jest do niego przypisana trasa, ma swój autobus z kierowcą. Ponadto kurs może zostać odwołany, gdy zapełnienie autobusu nie przekracza 15% wszystkich dostępnych miejsc. Aktualna lokalizacja X i Y autobusu odbywającego kurs będzie aktualizowana za pomocą urządzeń nawigacji.

Tabela 2-36 Opis encji "kurs" (źródło: opracowanie własne)

Nazwa encji	Kurs
Nazwa i opis atrybutów	Numer kursu - każdy kurs ma swój unikatowy identyfikator w formacie: kolejny następujący po sobie numer + skrót pierwszego przystanku + skrót ostatniego przystanku
	Numer trasy - jedna z tras będących w bazie danych
	Numer biletu - wszystkie bilety kupione na dany kurs

	<i>Numer kierowcy</i> - jest nadawany wczesnej i zapisywany do bazy, dany dostępny kierowca jest przypisany do danego kursu
	<i>Numer autobusu</i> - identyfikator autobus będącego w bazie przypisanego do danego kursu
	<i>Lokalizacja autobusu X</i> - realna współrzędna geograficzna X autobusu będącego w trasie
	<i>Lokalizacja autobusu Y</i> - realna współrzędna geograficzna Y autobusu będącego w trasie

Modelowanie struktury bazodanowej za pomocą encji pozwala na wizualne jej przedstawienie wraz z polami w przyszłych tabelach w bazie danych, oraz relacjami między nimi. Takie podejście ułatwia zrozumienie złożoności systemu, umożliwia identyfikację potencjalnych problemów na wczesnym etapie projektowania oraz stanowi fundament dla tworzenia efektywnych i zoptymalizowanych baz danych.

2.7. Modelowanie czynności i zakresu odpowiedzialności systemu

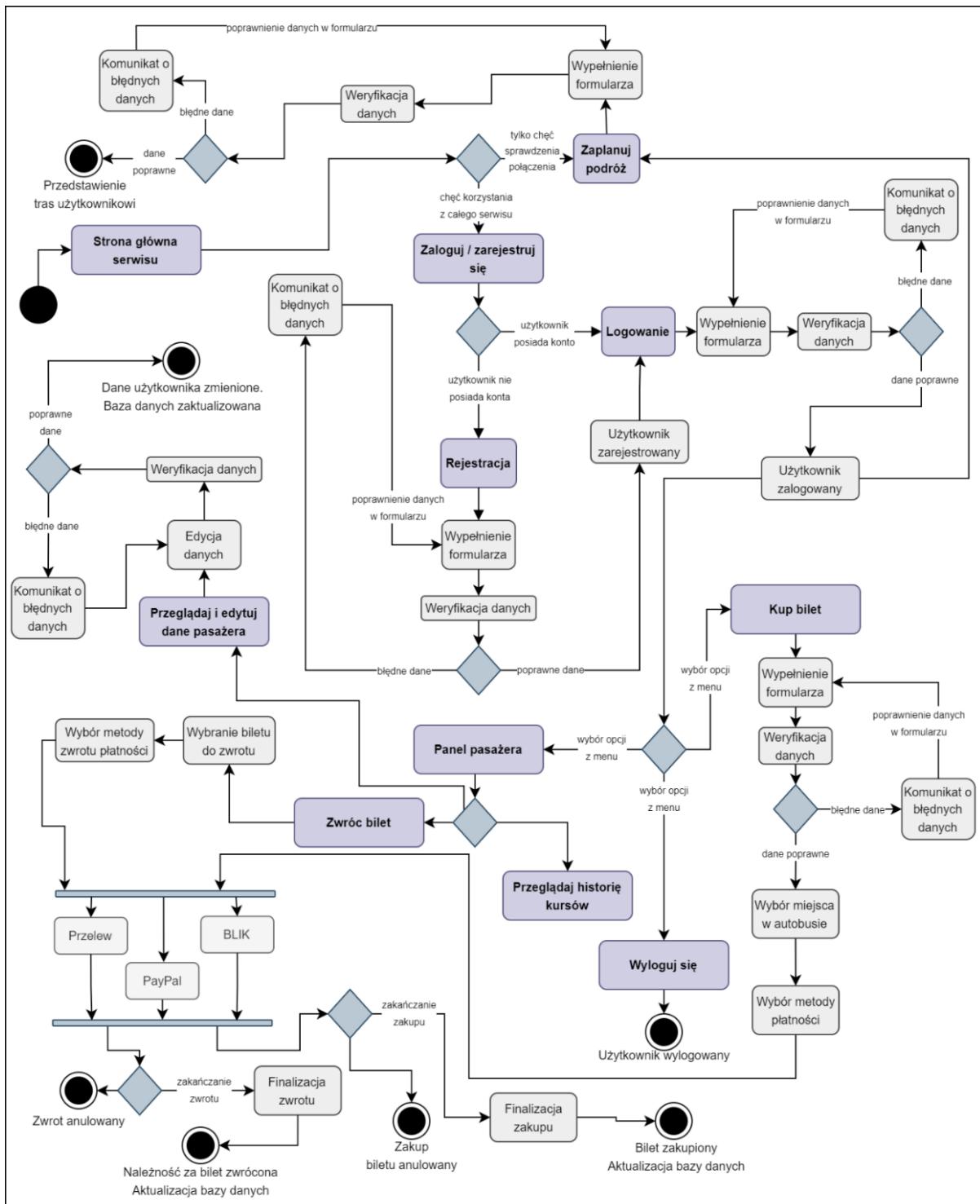
W kontekście projektowania systemów informatycznych istotnym aspektem jest modelowanie i udokumentowanie procesów w nim zachodzących. By to uczynić, budowanym systemie wykorzystano diagramy aktywności ukazujące jak system będzie reagował na zdarzenia.

Czarne koło stanowi węzeł początkowy, element rozpoczynający aktywność systemu. Reprezentuje początek przepływu aktywności i sterowanie procesami zachodzącymi w systemie. Węzłami końcowymi są czarne koła w okręgu, elementy kończące aktywność w systemie. Wtedy to działanie lub proces pomyślnie został wypełniony i dobiegł końca i nie należy podejmować żadnych dalszych działań ani kroków. Szaroniebieskie romby symbolizują decyzje podejmowane w systemie, gdzie określane są jej warunki. W zależności od warunku, przepływ może podążać różnymi ścieżkami. Fioletowe prostokąty z zaokrąglonymi rogami ukazują funkcjonalność dostępną z poziomu użytkownika obu aplikacji systemu. Szare prostokąty z zaokrąglonymi rogami pokazują procesy, jakie są wywoływanie w systemie po wcześniejszej interakcji i działaniu użytkownika. Kierunek przepływu ukazują odpowiednio skierowane czarne strzałki. Ich kierunek wskazuje kierunek przepływu aktywności w tworzonym systemie. [12]

2.7.1. Serwis internetowy dla pasażera

Funkcjonalności wraz z reakcjami wynikającymi z wchodzenia w interakcję z serwisem internetowym przedstawia rysunek „Diagram aktywności dla serwisu internetowego”.

Początkiem interakcji z serwisem internetowym będzie wejście na stronę przez użytkownika. Będzie on miał do wyboru trzy możliwości: zaplanowanie podróży, co umożliwia na przejrzenie i zapoznanie się z oferowanymi trasami przez przewoźnika, oraz zalogowanie się i zarejestrowanie nowego konta. Po udanym logowaniu, użytkownik dostaje dostęp do „panelu pasażera”, centrum stanowiącego zbiór informacji, dotychczasowych działań danego użytkownika.



Rysunek 2-4 Diagram aktywności dla serwisu internetowego (źródło: opracowanie własne)

Dopiero zalogowany użytkownik może zakupić bilet. Czyni to wchodząc w menu „kup bilet”, znajdujące się obok menu „panel pasażera”. Wywoływany jest wtedy proces mający doprowadzić do zakończenia procesu kupna biletu. System wyświetla formularz wymagający wypełnienia danymi pasażera, po czym użytkownik go wypełnia, jeżeli dane okażą się błędne, użytkownik zostanie o tym poinformowany. Po poprawnym wypełnieniu formularza system przejdzie do kolejnego kroku, wyboru miejsca w autobusie. Pasażer wybiera jedno miejsce. Następnie system pokaże okno umożliwiające wybór jednej z kilku dostępnych u danego przewoźnika metod płatności. W ostatnim kroku użytkownik ujrzy podsumowanie zamówienia wraz ze szczegółami przejazdu. Zaistnieje wtedy możliwość anulowania całego zamówienia lub jego końcowej finalizacji, zaakceptowania i zapłaty wybraną wcześniej metodą płatności. Następuje zakończenie całego procesu, system zapisuje dane w bazie danych i generuje bilet.

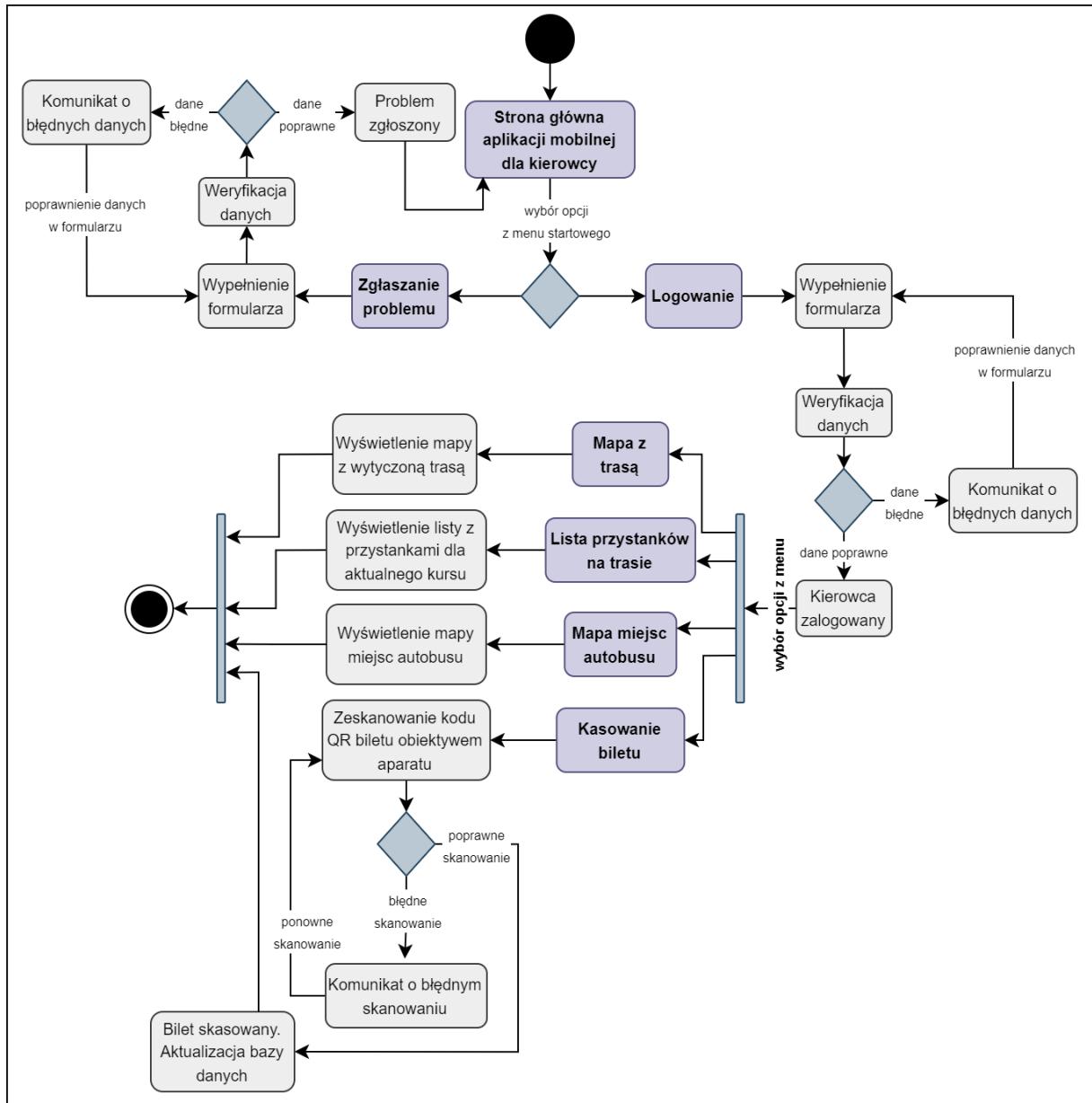
Wcześniej wspomniany „panel pasażera” umożliwi zalogowanemu użytkownikowi dostęp do dwóch opcji: zwrotu wcześniej kupionego biletu i przeglądania dotychczasowych, wcześniej odbytych przejazdów. W drugim z wymienionych działań system generuje listę z danymi dotyczącymi wcześniej odbytych przejazdów. Wybierając pierwszy z nich system wygeneruje, jeżeli będzie to możliwe, listę z biletami podlegającymi zwrotowi. Użytkownik wybierze jeden z nich, po czym system poprosi o sprecyzowanie metody zwrotu należności za bilet. Końcowa decyzja systemu, wynikająca ze wcześniejszej interakcji użytkownika z serwisem, przerwie lub sfinalizuje i zakończy proces zwrotu z sukcesem i zaktualizuje bazę danych.

Użytkownik zalogowany z menu „panel pasażera” może też wybrać opcję „przeglądaj i edytuj dane pasażera”. System rozpocznie nowy proces. Zależnie od dalszych decyzji użytkownika, zostaną mu umożliwione aktywności edycji swoich danych wprowadzonych wcześniej w formularzu rejestracji oraz zmiany hasła. W obu przypadkach, kiedy dane zostaną wprowadzone, system je zweryfikuje. Jeżeli zostanie stwierdzony błąd we wprowadzanych danych, użytkownik zostanie poproszony o ich poprawienie. W przypadku stwierdzenia pełnej poprawności danych system finalizuje proces i aktualizuje bazę danych.

2.7.2. Aplikacja mobilna dla kierowcy

Przepływ aktywności w aplikacji graficznej przedstawia rysunek „Diagram aktywności dla aplikacji mobilnej”. Kierowca na swoim urządzeniu uruchamia aplikację, co stanowi początek aktywności drugiego komponentu budowanego systemu. Po uruchomieniu system wyświetla okno z dwoma możliwymi działaniami do podjęcia: zalogowania i zgłoszenia problemu. Kierowca, mając wcześniej założone konto, uruchamia proces logowania się poprzez

dotknięcie odpowiedniej opcji na ekranie. Wpisuje wymagane dane, które następnie system zweryfikuje. Po nieudanej weryfikacji kierowcy zostanie wyświetlony odpowiedni komunikat, zaś po udanej system umożliwi kierowcy z korzystania z niego w pełni.



Rysunek 2-5 Diagram aktywności dla aplikacji mobilnej (źródło: opracowanie własne)

Będąc jeszcze na ekranie początkowym aplikacji kierowca może zgłosić dowolny napotkany problem. Dotyka przycisk zlokalizowany pod przyciskiem odpowiedzialnym za zalogowanie. Użytkownikowi ukaże się formularz z kilkoma polami do wypełnienia. System wypełnione pole zweryfikuje, w przypadku stwierdzenia błędu, kierowcy ukaże się odpowiedni komunikat. Po pomyślnej weryfikacji system przekaże formularz dalej i zaktualizuje bazę danych o jego treść. Po zakończeniu procesu system przejdzie do ekranu głównego.

Wcześniej zalogowany kierowca uzyskuje dostęp do czterech funkcji aplikacji, pomiędzy którymi może się dowolnie przełączać:

- Mapy z aktualnie odbywaną trasą – wyświetlanie interaktywnej mapy z zaznaczoną aktualnie odbywaną trasą.
- Listy z przystankami na trasie – listy z danymi o przyjazdach i odjazdach na przystankach wcześniej zaplanowanej trasy.
- Mapy z rozmieszczeniem i zajętością miejsc w autobusie – informacja dla kierowcy, jakie miejsca zostały przydzielone pasażerom po skasowaniu im biletów.
- Kasowania biletu – proces uruchamiający kamerę urządzenia używanego przez kierowcę autobusu. Po ukazaniu się obrazu z aparatu kierowca skierowuje go na kod QR biletu. System rozpoznaje go i weryfikuje jego poprawność. Po nieudanej weryfikacji w oknie widoku aparatu pojawi się stosowny komunikat. Po udanej weryfikacji bilet zostanie skasowany, co oznacza przypisanie wcześniej wybranego miejsca pasażerowi w autobusie, zapisanie danych w bazie danych i jej aktualizację, stanowi to zakończenie procesu kasowania biletu.

Każda z czterech wymienionych wyżej funkcjonalności ma wspólny węzeł końcowy. Każdorazowo, po wejściu w interakcję z którąś z nich istnieje możliwość przejścia do innej funkcjonalności aplikacji.

2.8. Bezpieczeństwo i ochrona danych

W celu zapewnienia bezpieczeństwa danych w budowanym systemie informatycznym przewiduje się uwzględnienie odpowiednich do sytuacji zasad, procedur, praktyk i technologii. Odpowiednio wdrożone i przestrzegane zminimalizują ryzyko wystąpienia incydentów związanych z nieautoryzowanym dostępem do danych, ich utratą czy naruszeniem poufności, np.: dane dotyczące płatności.

Wdrożenie oraz przestrzeganie mechanizmów identyfikacji i uwierzytelniania użytkowników zapobiegną nieautoryzowanym dostępom do systemu. Ich elementem może być polityka hasłowa: wymóg stosowania silnych haseł i cyklicznych ich zmian.

Najwrażliwsze dane (numery kart płatniczych pasażerów, dane dotyczące dokumentów kierowców uprawniające ich do jazdy) powinny być szyfrowane i zabezpieczone przed nieautoryzowanym dostępem podczas ich przechowywania, przesyłania i przetwarzania.

Szyfrowanie powinno być stosowane zarówno na poziomie komunikacji sieciowej, jak i po stronie aplikacji czy rozwiązań odpowiedzialnych za przechowywanie danych.

Dostęp do systemu informatycznego oparty na rolach polega na przydzieleniu użytkownikom określonych uprawnień i zasobów w oparciu o ich funkcje lub zadania w organizacji. Każda rola ma przypisane określone uprawnienia, co umożliwia zarządzanie dostępem w sposób bardziej zorganizowany i bezpieczny. Użytkownicy otrzymują dostęp tylko do tych zasobów, które są niezbędne do wykonywania ich obowiązków, co minimalizuje ryzyko naruszeń bezpieczeństwa i nadużyć.

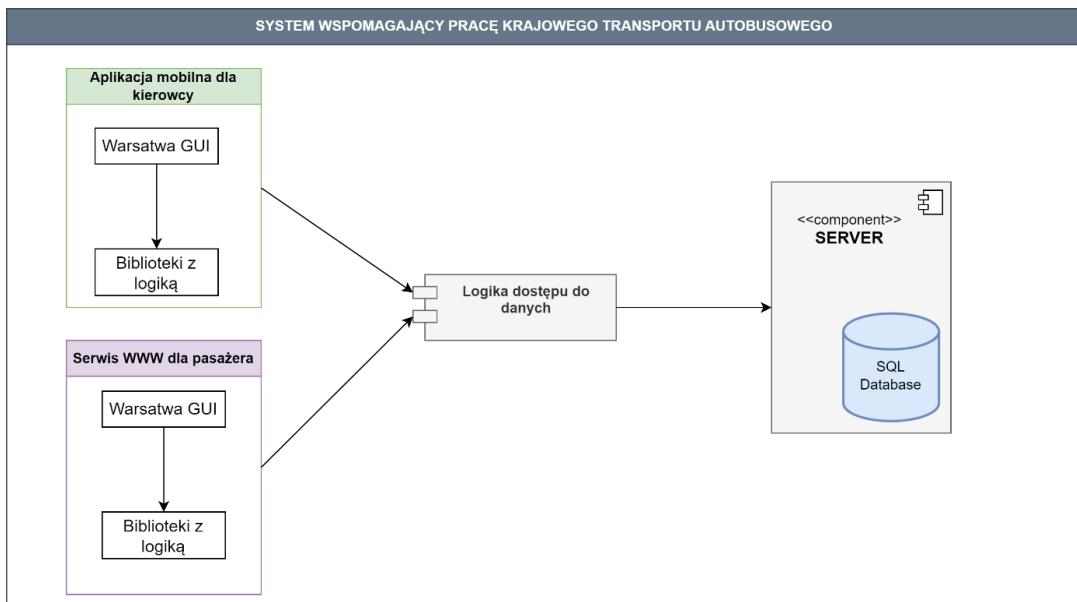
Pojawienie się incydentu w postaci nieautoryzowanego działania z zewnątrz, utraty danych czy naruszenia poufności powinno zostać odpowiednio monitorowane i raportowane. Wcześniej wprowadzona i przestrzegana polityka bezpieczeństwa określi zasady i działania, jakie należy podjąć w przypadku wystąpienia incydentu, a także pozwoli wyciągnąć wnioski i analizy, by poprawić bezpieczeństwo całego systemu informatycznego w przyszłości.

Wdrażanie polityk bezpieczeństwa w systemach informatycznych jest rzeczą kluczową dla ochrony danych każdego z pasażerów i pracowników przedsiębiorstw korzystających z rozwiązania informatycznego, ale też często jest wymagane prawnie. Skuteczne opieranie się zewnętrznym, niepożądanym działaniom pomaga zwiększyć zaufanie i lojalność klientów, oraz pokazuje przedsiębiorstwo w pozytywnym świetle, bo dba o prywatność i poufność danych, które są jemu powierzane. [13]

2.9. Wstępna architektura systemu

System ma działać na wielu urządzeniach jednocześnie, więc konieczne będzie użycie metod programowania rozporoszonego. Jest to podejście łączące ze sobą co najmniej dwa systemy. Podejście takie zawiera w sobie tzw. middleware - oprogramowanie pośredniczące między modułami aplikacji. Biblioteki wykorzystywane w programowaniu rozproszonym umożliwiają aplikacji serwerowej na wysyłanie komunikatów do podłączonych.

Schemat przedstawia wstępную architekturę budowanego systemu. Dwa komponenty, serwis internetowy i aplikacja dla kierowcy, komunikować się będą z serwerem baz danych poprzez biblioteki z logiką. Odpowiednio skonfigurowana warstwa logiki pozwoli na obustronną komunikację pomiędzy serwerem a wielomainstancjami aplikacji.



Rysunek 2-6 Wstępny schemat architektury systemu (źródło: opracowanie własne)

Logika odpowiedzialna za dostęp oraz operacje na danych będzie kluczowa w procesie zakupu biletów. Wielu użytkowników w tym samym czasie może dokonywać zakupu. System musi powiadomić użytkownika o miejscach czasowo zarezerwowanych na poczet innych użytkowników będących w trakcie procesu zakupu biletów. [12]

3. PROJEKT

Na samym początku rozdział poświęcony projektowi budowanego rozwiązania informatycznego skupi się na wstępny przedstawieniu i omówieniu wykorzystywanych technologii i narzędzi. Następnie, w kolejnych dwóch podrozdziałach, szczegółowo przedstawiany jest projekt interfejsu użytkownika dla serwisu internetowego i aplikacji mobilnej. Na jego podstawie została doprecyzowana funkcjonalność systemu, wcześniej omówiona w rozdziale dotyczącym analizy tworzonego systemu. Całość projektu interfejsu sporządzono w programie do edycji grafiki wektorowej – CorelDraw Graphic Suite 2022. Grafiki dotyczących interfejsu użytkownika mają odzwierciedlać rozłożenie kontrolek na stronie internetowej i w aplikacji dla kierowcy, co umożliwi pełne poznanie funkcjonalności budowanego systemu. Paleta kolorów w końcowym systemie będzie się różnić od tej stosowanej w tym rozdziale, zgodnie ze wcześniej precyzowanymi wymaganymi pozafunkcjonalnymi.

Czwarty podrozdział preczyje i przybliża strukturę bazodanową. Omówiony został projekt bazy danych w postaci diagramu z tabelami odpowiadającymi tabelom w bazie wraz z ograniczeniami, typami danych i relacjami między nimi.

Jako ostatnią część rozdziału pracy inżynierskiej traktującego o projekcie zostało sprecyzowanie i omówienie architektury całego budowanego rozwiązania. Omówiono w nim zastosowane podejścia jakie zostaną wykorzystane w trakcie budowania systemu wraz z komponentami i założeniami całej architektury.

3.1. Wybrane technologie

System wspomagający pracę krajowego transportu autobusowego zostanie stworzony w całości w jednej platformie programistycznej - Microsoft .NET. będącej zestawem narzędzi, bibliotek umożliwiających tworzenie, wdrażanie i zarządzanie różnorodnymi aplikacjami. .NET opiera się na zorientowanym obiektowo języku programowania C#, ale obsługuje również języki takie jak: Visual Basic, F#, C++ jak i inne. Dostarcza również wiele struktur, bibliotek, gotowych narzędzi do budowania aplikacji, usług lub innych projektów programistycznych ujętych we „frameworki” - zbiory szkieletów i reguł do budowy aplikacji.

Cały system będzie tworzony w środowiskach programistycznych, IDE - Integrated Development Environment, stanowiących zestaw narzędzi i funkcji, które ułatwiają pisanie i testowanie kodu źródłowego. W toku implementacji systemu zostaną wykorzystane głównie dwa środowiska programistyczne: Visual Studio 2022, głównie do działań związanych z

wymianą danych w systemie, czy też ich autoryzacją i obsługą błędów, oraz Visual Studio Code do czynności związanych z interfejsem użytkownika.

Do stworzenia systemu informatycznego wspomagającego krajowy transport autobusowy zostaną wykorzystane następujące technologie i narzędzia:



Rysunek 3-1 Logotypy wykorzystywanych środowisk programistycznych. (źródło: <https://visualstudio.microsoft.com/pl/free-developer-offers>)

- a) Microsoft SQL Server – serwer bazy danych, obecnie jeden z najpopularniejszych silników bazodanowych. Do stworzenia systemu został wybrany relacyjny model oparty na tabelach, co zapewni klarowność danych. Poprawność implementowanej bazy danych będzie sprawdzana w SQL Server Management Studio, środowisku umożliwiającym zarządzanieinstancjami serwera SQL.
- b) Entity Framework - narzędzie ORM (Object-Relational Mapping), które ułatwia mapowanie obiektów do struktury baz danych na podstawie wcześniej utworzonych klas w języku programowania C#.
- c) C# - obiektowy język programowania stworzony przez Microsoft. Jest językiem typowanym, każdy typ zmiennej musi być wcześniej jawnie określony. Obecnie jest używany jako wiodący język platformy .NET. Możliwe jest też tworzenie gier za pomocą silnika Unity obsługującego C# jako jeden z języków programowania. Obiektywo zorientowanie pozwala na używanie technik dziedziczenia, polimorfizmu, enkapsulacji czy abstrakcji.
- d) ASP. NET MVC – framework wykorzystywany do zbudowania serwisu internetowego. Stworzony przez Microsoft i wykorzystywany do budowania aplikacji przeglądarkowych. Wykorzystuje wzorzec MVC – model – widok (view) – kontroler (controller). W tej technologii zostaną wykorzystane inne, pomniejsze:
 - HTML (Hypertext Markup Language) - język znaczników używany do tworzenia struktur strony internetowej. Podstawowe narzędzie budowania i definiowania struktury stron internetowych. Składa się z zestawu znaczników

definiujących różne elementy strony, takie jak nagłówki, paragrafy, linki czy obrazy. Jeden z trzech elementów budowania współczesnych stron (poza CSS i JavaScript)

- CSS, Bootstrap (Cascading Style Sheets) - język służący do stylizacji dokumentów HTML. Pozwala na definiowanie wyglądu i układu elementów na stronie internetowej. Oddziela warstwę prezentacji od struktury dokumentu, co pozwala na zastosowanie jednego zestawu stylów dla wielu stron. W pliku odpowiedzialnym za style definiowane są reguły składające się z selektorów i deklaracji. Selektory wskazują, które elementy HTML mają być stylizowane, a deklaracje określają właściwości takie jak kolor, rozmiar czcionki, odstępy czy tło. Istnieje również koncepcja kaskadowania, co oznacza, że style mogą być dziedziczone z poziomu nadzędnegoo do podzędnegiego elementu. Bootstrap zaś jest frameworkm CSS. Oferuje gotowe komponenty służące tworzeniu interfejsu użytkownika.
 - JavaScript – wszechstronny język programowania początkowo stworzony do obsługi interaktywności na stronach internetowych. Jego dynamiczne typowanie, funkcyjne podejście do tworzenia kodu oraz obsługa asynchroniczności sprawiają, że jest elastyczny i efektywny w obszarze tworzenie stron i aplikacji internetowych. Bogactwo dostępnych bibliotek i frameworków pozwala na rozwijanie kompletnych aplikacji zarówno po stronie klienta, jak i serwera, oraz umożliwia rozwijanie zaawansowanych i skalowalnych projektów webowych.
- e) .NET MAUI (Multi-platform App UI) – framework wykorzystany do zbudowania aplikacji mobilnej dla kierowcy. Umożliwia tworzenie aplikacji wieloplatformowych. Kod aplikacji jest wspólny dla systemów: Windows, Android i iOS, co umożliwia używania jednego zestawu narzędzi, języka i struktury projektu do budowy aplikacji na różnych platformach, co skraca czas rozwoju i ułatwia utrzymanie. Za warstwę logiki aplikacji odpowiada język programowania C#, za prezentacji, XAML.
- XAML - (Extensible Application Markup Language) - deklaratywny język znaczników (podobny do HTML) używany głównie do definiowania struktury interfejsów użytkownika w aplikacjach opartych na platformie Microsoftu. Ułatwia tworzenie czytelnych struktur interfejsu korzystając z hierarchii znaczników. Poszczególne elementy interfejsu, takie jak przyciski, pola tekstowe czy panele, są definiowane przy użyciu odpowiednich znaczników, a ich właściwości są ustawiane w atrybutach używanych tagów.

Szczegółowe omówienie sposobów zastosowania poszczególnych wykorzystywanych technologii i narzędzi znajduje się w rozdziale dedykowanym implementacji budowanego systemu. [14] [15]

3.2. Projekt interfejsu użytkownika dla serwisu internetowego

3.2.1. Strona główna serwisu

Stronę domową serwisu internetowego dla pasażera ukazuje rysunek „Projekt widoku strony domowej serwisu”. Przedstawia on treści ogólne dotyczące aktualności, rabatów na przejazdy, czy najważniejsze komunikaty i ogłoszenia. Za nawigację po niej będzie odpowiadać, widoczna w trakcie przewijania, górną belką z trzema zakładkami. W lewym górnym rogu znajdzie się logo, odpowiednie dla podmiotu mającego wdrożoną, wcześniej dopasowaną do wymagań stronę. Kliknięcie w nie przekierowuje użytkownika na stronę główną z wyżej wymienioną ogólną treścią.



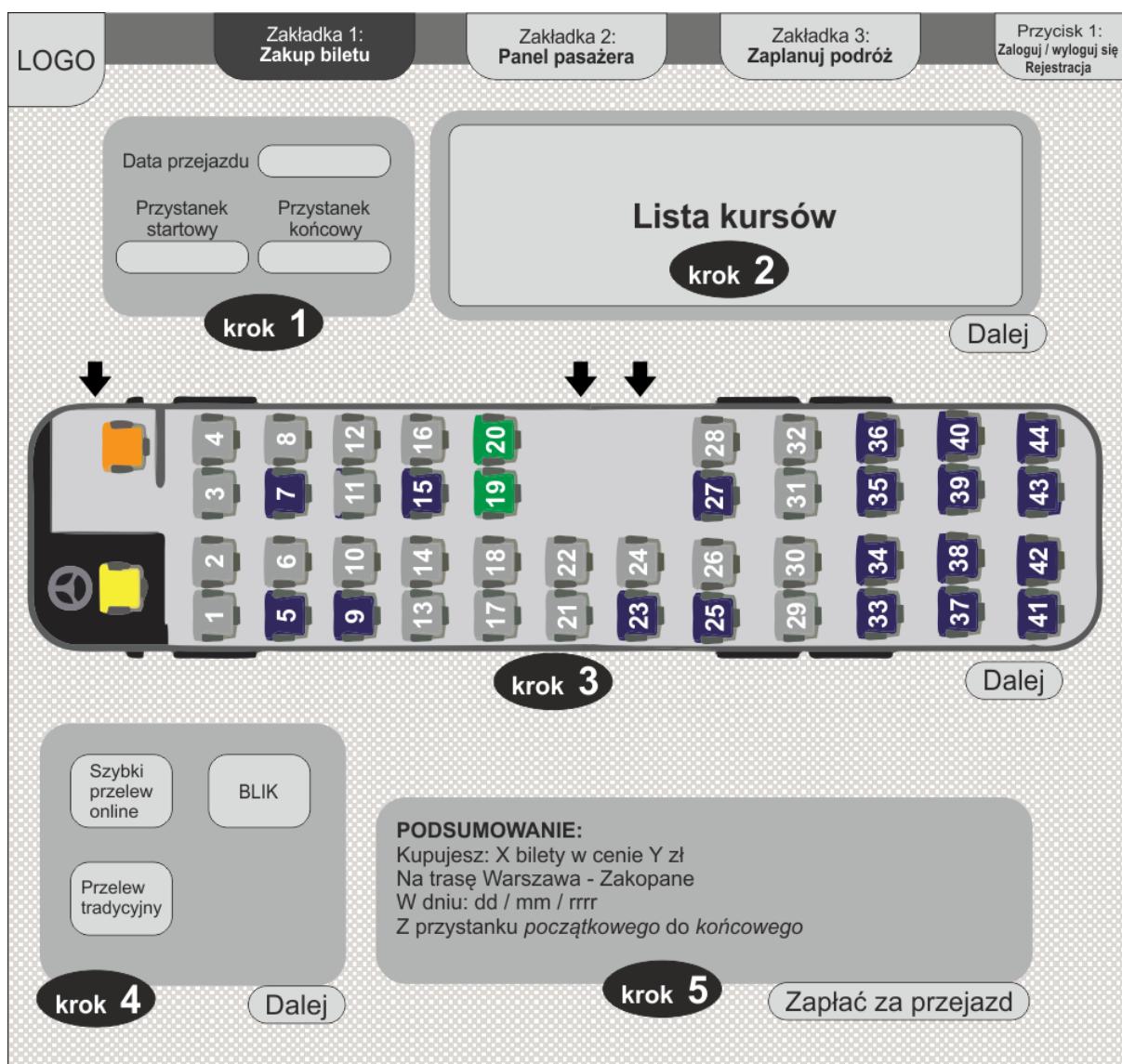
Rysunek 3-2 Projekt widoku strony domowej serwisu (źródło: opracowanie własne)

Pierwsza zakładka wywoła menu zakupu biletu, druga pozwoli na wejście do panelu pasażera. Obie te zakładki będą dostępne tylko po zalogowaniu się na wcześniej utworzone konto. Trzecia zakładka, „zaplanuj podróż”, pozwala na przeglądanie dostępnych tras z poziomu zarówno zalogowanego, jak i niezalogowanego użytkownika. Przeznaczeniem tej podstrony jest przedstawienie potencjalnemu pasażerowi oferty przejazdów, jakie dana firma może zaproponować. Przycisk w prawym górnym rogu serwisu, w zależności od kontekstu, pozwoli na rejestrację, zalogowanie bądź wylogowanie się z serwisu.

3.2.2. Zakładka 1: Zakup biletu

Zakładka „Zakup biletu”, ukazana w rysunek „Projekt widoku strony "zakup biletu"" krok po kroku przeprowadza pasażera przez proces kupna biletu. By z niej skorzystać użytkownik musi być zalogowany do serwisu. Zakładka będzie widoczna dla użytkownika niezalogowanego, lecz niedostępna będzie jej funkcjonalność. Wchodząc w nią wyświetli się okno z informacją o możliwości logowania i rejestracji wraz z linkami przekierowującymi do tych funkcjonalności.

Tuż po wejściu w zakładkę widoczna będzie tylko część całego formularza – ta odpowiadająca za krok pierwszy w procesie zakupu biletu.



Rysunek 3-3 Projekt widoku strony "zakup biletu" (źródło: opracowanie własne)

W nim, użytkownik określa datę przejazdu i wypełnia pola „przystanek początkowy” i „przystanek końcowy”. Po pomyślniej weryfikacji danych przez system, po prawej stronie

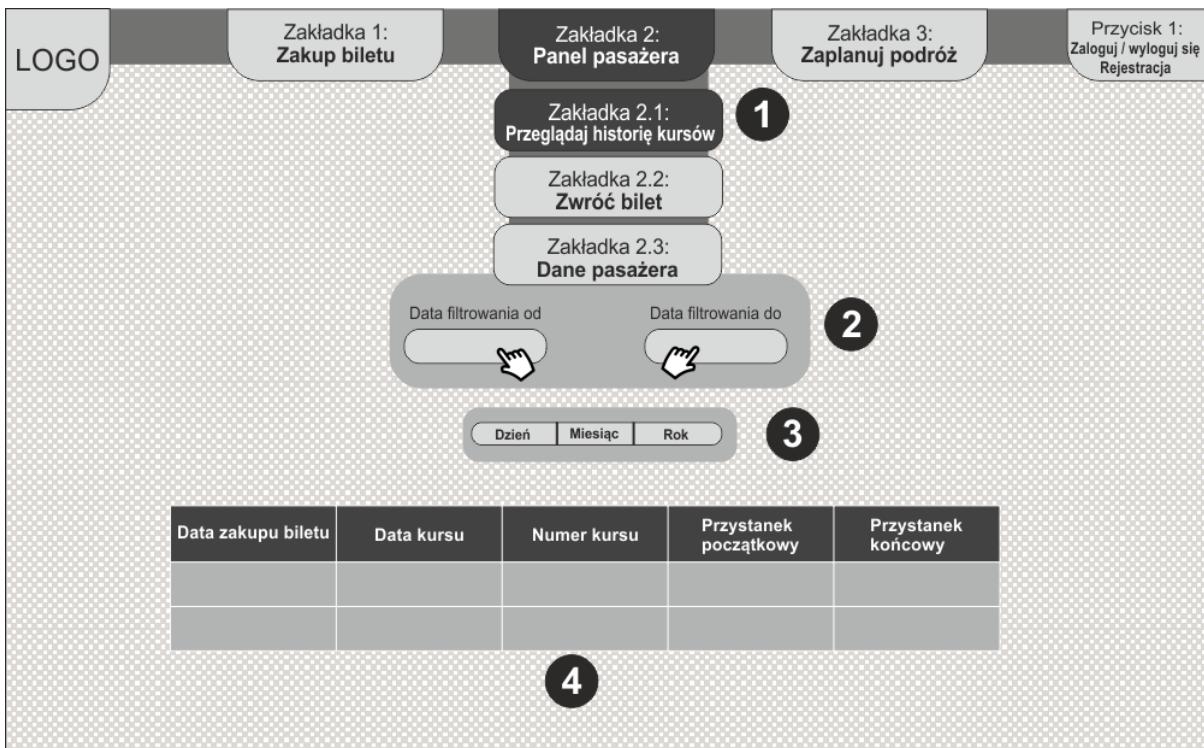
pierwszej części formularza wyświetli się lista kursów zgodnych ze wcześniej określonymi kryteriami (krok 2 na obrazku). Po wyborze kursu z listy w kroku drugim, jako krok trzeci, wygeneruje się mapa autobusu. Wybierając pola zaznaczone kolorem szarym, odpowiedzialne za wizualizację wolnych miejsc, użytkownik podświetla je na zielono – to do tych miejsc zostaną wygenerowane bilety w późniejszych krokach procedury zakupu. Niebieskie pola obrazują miejsca już zajęte. Po pomyślnym wyborze miejsc siedzących system wygeneruje kolejną część formularza odpowiedzialnego za wybór metody płatności – krok czwarty. Użytkownik wybierając jedną z kilku dostępnych i naciskając przycisk „dalej” przechodzi do ostatniego, piątego kroku w formularzu – podsumowania całego procesu, gdzie przedstawione będą szczegóły wcześniejszych wyborów użytkownika, takie jak: całkowita wartość biletów na przejazd, jego data i trasa jaką przebiega.

Dla użytkownika niezalogowanego zakładka będzie widoczna, lecz ten nie będzie mógł wejść z nią w żadną interakcję. Po próbie wejścia w tę zakładkę użytkownik zostanie poinformowany wyskakującym okienkiem o możliwości zalogowania, lub założenia nowego konta. Do obu tych opcji zostaną

3.2.3. Zakładka 2: Panel pasażera

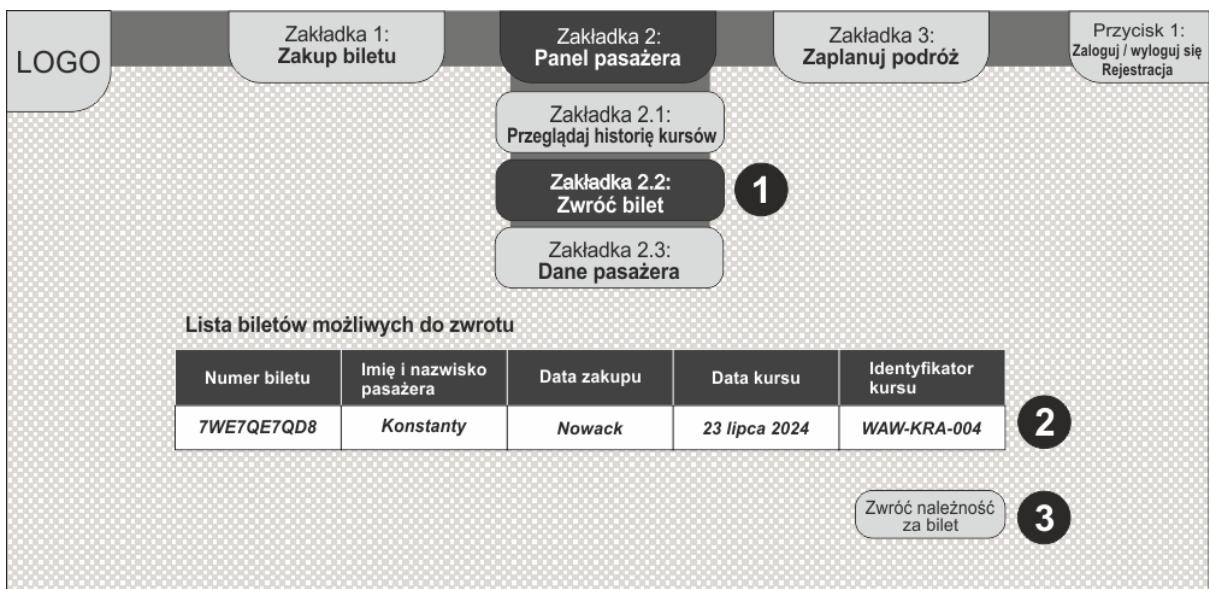
Centralną częścią funkcjonalności serwisu dla zalogowanego użytkownika jest druga zakładka „panel pasażera”. Zawiera ona w sobie dwa podmenu ukazujące się po najechaniu za zakładkę i kliknięciu w nią: „przeglądaj historię kursów” oraz „zwróć bilet”. Użytkownik nie wybrał jeszcze żadnego podmenu będzie widział ostatnią odwiedzaną przez siebie stronę z serwisu. Rozwijające się menu znika po kliknięciu w jedną z opcji, lub w dowolny obszar strony serwisu.

Zakładka 2.1, ukazana w rysunku „Projekt widoku strony "Przeglądaj historię kursów"" odpowiada za przeglądanie odbytych, będących w realizacji lub aktualnie w toku kursów. Po wejściu w nią użytkownikowi wyświetla się dwa pola tekstowe z „datą filtrowania od” i „datą filtrowania do” (punkt 2 na schemacie interfejsu) służące zawężeniu wyszukiwania spośród wszystkich jego kursów.



Rysunek 3-4 Projekt widoku strony "Przeglądaj historię kursów" (źródło: opracowanie własne)

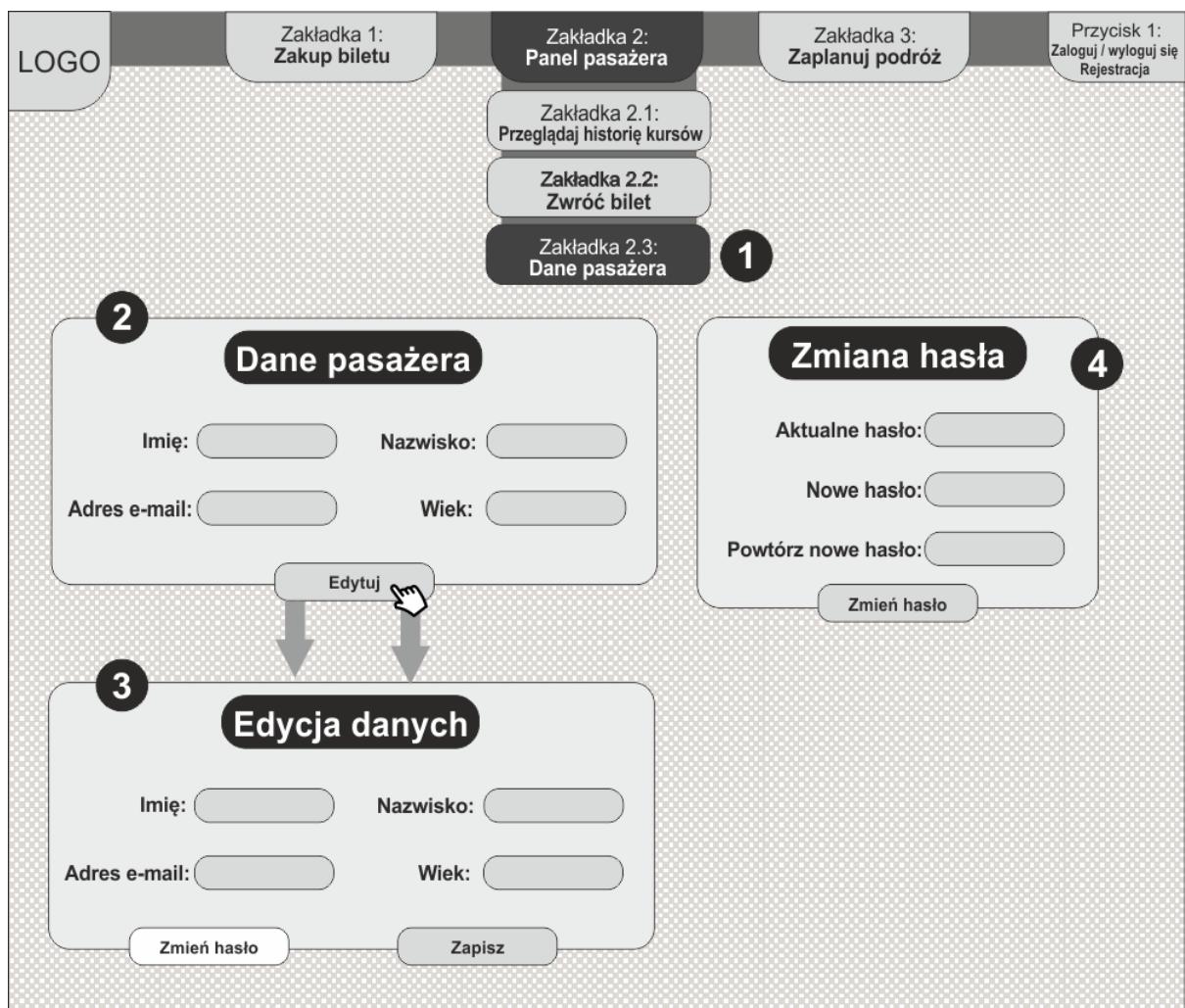
Użytkownik klikając w jedno z pól odpowiadających za zakres filtrowania wywoła małe wyskakujące okno pozwalające na sprecyzowanie interesującej go daty (punkt 3 na schemacie interfejsu). Jeżeli pola zostaną pozostawione pustymi, w wygenerowanej tabeli będzie widnieć jedynie ostatnie 10 najnowszych przejazdów (punkt 4 na schemacie interfejsu).



Rysunek 3-5 Projekt widoku strony "Zwróć bilet" (źródło: opracowanie własne)

Przedostatnia pozycja w menu panelu pasażera, ukazana w rysunku „Projekt widoku strony "Zwróć bilet"” wywołuje okno odpowiadające za proces zwrotu biletu. Dostęp do niej,

jak i do funkcjonalności całego panelu pasażera, będzie miał wyłącznie użytkownik zalogowany.



Rysunek 3-6 Projekt widoku strony "Dane Pasażera" (źródło: opracowanie własne)

Po wejściu w zakładkę ukaże się tabela ze wszystkimi możliwymi do zwrotu biletami (punkt 2 na obrazku Zakładka 2.2: Zwróć bilet). Użytkownik wybiera jeden z nich i przechodzi dalej, do wybranej metody zwrotu należności z kilku aktualnie dostępnych, naciskając przycisk „zwróć należność za bilet”.

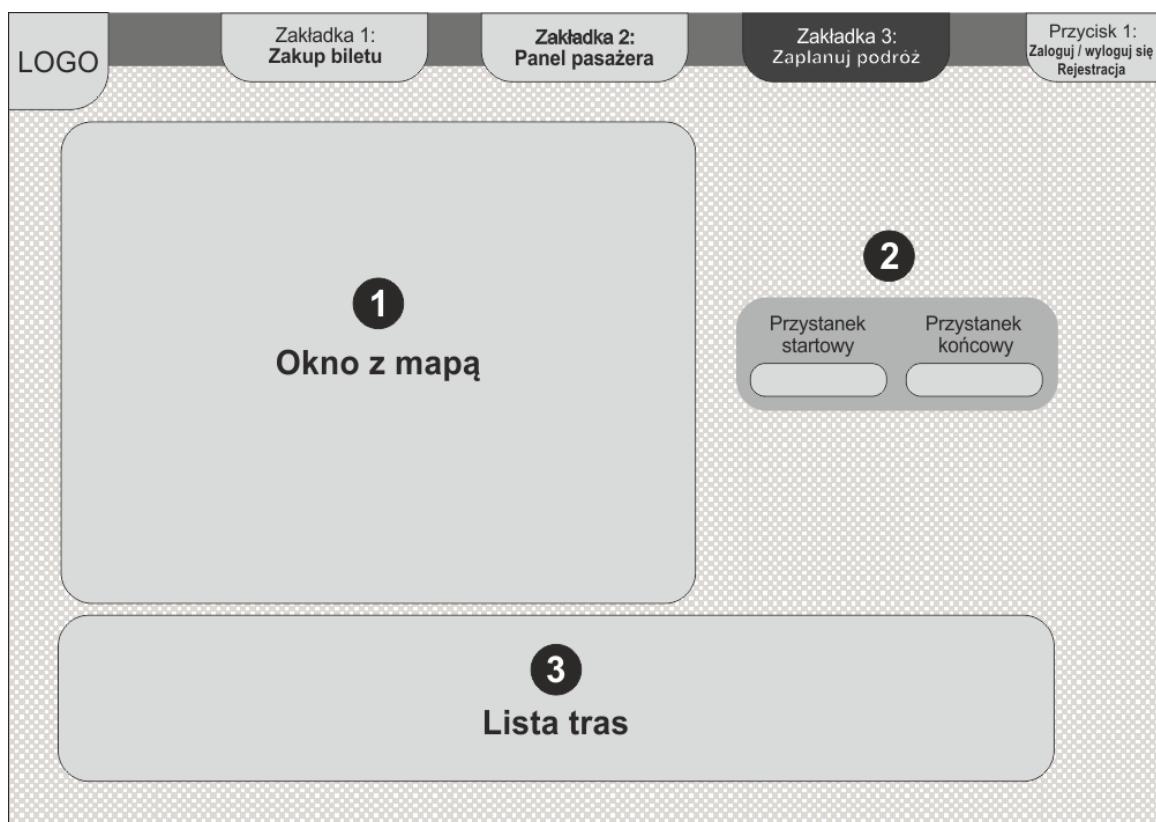
Pozycja będąca na samym dole zakładki „Panel pasażera” wyświetli dane pasażera. Początkowy widok nie będzie pozwalał na ich edycję, to dopiero umożliwi wejście w interakcję z kolejnym elementem tego widoku – polem do edycji danych wywoływanym naciśnięciem przycisku „Edytuj” pod tymi danymi. Wszystkie pola w formularzu zostaną odblokowane do edycji, po wpisaniu nowych danych system zweryfikuje ich poprawność, co zakomunikuje odpowiednim powiadomieniem. Pojawi się też nowa opcja – „Zmień hasło”, działa on analogicznie do zmiany danych pasażera, z tym, że weryfikacja jego poprawności będzie się

odbywać na podstawie wdrożonej przez przewoźnika polityki bezpieczeństwa dotyczącej haseł, o czym użytkownik zostanie poinformowany odpowiednim komunikatem.

3.2.4. Zakładka 3: Zaplanuj podróż

Trzecia zakładka jako jedyna będzie dostępna dla pasażera niezalogowanego. Ukazuje ją rysunek "Projekt widoku strony "Zaplanuj podróż"". Umożliwi ona przeglądanie dostępnych tras na podstawie określonego przystanku początkowego i końcowego, oraz planowanie przejazdów przez zainteresowanego.

Główną częścią okna będzie komponent z mapą. Pole to będzie w pełni interaktywne i umożliwi, poprzez kliknięcie na mapie, ustawienie dwóch wskaźników na niej, z potencjalnym początkiem i końcem trasy. System samodzielnie wyszuka najbliższe przystanki odpowiadające wyborom użytkownika i je zaproponuje jako punkt początkowy i końcowy trasy.



Rysunek 3-7 Projekt widoku strony "Zaplanuj podróż" (źródło: opracowanie własne)

Inną możliwością na określenie początku i końca potencjalnej podróży jest wpisanie nazw miejscowości punkcie drugim, w polach „przystanek startowy” i „przystanek końcowy”. System nanieje punkty na mapę i wyświetli (w punkcie trzecim na obrazku „Zakładka 3:

Zaplanuj trasę") potencjalne trasy przebiegające przez wskazanie miejscowości, lub zaproponuje przystanki leżące nieopodal tychże miejscowości.

3.2.5. Przycisk 1: Logowanie, wylogowanie i rejestracja

Zależnie od aktualnego kontekstu, „przycisk 1” może odpowiadać za jedną z trzech funkcjonalności, jago naciśnięcie wyświetli okno odpowiedzialne za:

- zalogowanie się klienta - pasażera, który wcześniej przeszedł cały proces rejestracji konta w serwisie
- wylogowanie się z serwisu,
- założenie nowego konta w serwisie.

The diagram illustrates a user interface design for a travel service website. At the top, there is a navigation bar with four tabs: 'Zakładka 1: Zakup biletu', 'Zakładka 2: Panel pasażera', 'Zakładka 3: Zaplanuj podróż', and a button labeled 'Przycisk 1: Zaloguj / wyloguj się Rejestracja'. The main content area is divided into two sections: '1: Logowanie' (top) and '2: Rejestracja' (bottom). The 'Logowanie' section contains fields for 'Adres e-mail:' and 'Hasło:', followed by a 'Zaloguj się' button and a link 'Nie masz konta? Zarejestruj się' which is being pointed to by a mouse cursor. The 'Rejestracja' section contains fields for 'Imię:', 'Nazwisko:', 'Wiek:', 'Adres e-mail:', 'Powtóż adres e-mail:', 'Hasło:', and 'Powtóż hasło:', followed by a 'Zarejestruj się' button. Arrows indicate a flow from the registration section back up to the login section.

Rysunek 3-8 Projekt widoku strony "Zaloguj/Wyloguj się. Rejestracja" (źródło: opracowanie własne)

Użytkownikowi niezalogowanemu wchodzącemu w interakcję z „przyciskiem 1” wyświetli się okno logowania (punkt 1: logowanie na rysunku: „Przycisk 1: zaloguj / wyloguj się, zarejestruj”). Wpisując poprawne dane w oknie logowania w polach „adres email”, oraz „hasło” użytkownik przejdzie do strony głównej serwisu.

Nie mając jeszcze konta, użytkownik może nacisnąć link „Nie masz konta? Zarejestruj się” by przejść do formularza rejestracji. Ten, w tym samym oknie podmieni się za okno logowania. Po wpisaniu poprawnych danych i zakończeniu reszty wymaganych czynności do założenia konta w serwisie użytkownik będzie mógł się zalogować i w pełni korzystać z całej funkcjonalności oferowanej przez serwis.

3.3. Projekt interfejsu użytkownika dla aplikacji mobilnej

Interfejs graficzny aplikacji mobilnej dla kierowcy przed wszystkim będzie dostosowany do wygodnej obsługi dotykiem. Główną częścią odpowiedzialną za nawigację po systemie będzie dolna belka z czterema kontrolkami mającymi za zadanie wyświetlenie:

- bieżącej trasy autobusu wraz z jego lokalizacją obliczaną i wyświetlana w czasie rzeczywistym
- mapy autobusu z rozmieszczeniem siedzeń i informacją o ich zajęciu,
- listy z przystankami znajdującymi się na aktualnym kursie autobusu,
- okna aparatu z funkcjonalnością kasowania biletów.

Ważną cechą budowanego interfejsu dla aplikacji będzie jego czytelność podczas jazdy. Zostaną użyte kontrastujące barwy, co pozwoli na odróżnienie treści, nawet w pełnym słońcu, która kontrolka, lub które okno aktualnie jest w użyciu. Opcja będąca aktualnie w użyciu zmieni na kolor wyraźnie ciemniejszy od pozostałych, tekst w niej zawarty zamieni kolor na biały, lub odcień lekkiej szarości. Dotykając kolejnych, innych kontrolek, wcześniej używana zmieni swój kolor na domyślny, jasny, a wybrana zmieni swoje barwy na ciemniejsze.

3.3.1. Logowanie i zgłaszanie problemu

Pierwszy ekran, jaki będzie widoczny po uruchomieniu aplikacji mobilnej przedstawia rysunek „Projekt okna logowania do aplikacji”. Kierowca, któremu wcześniej założono konto loguje się swoimi danymi wypełniając pole „numer kierowcy” oraz „hasło” i dotyka przycisk „zaloguj się”.

Logowanie do aplikacji

Numer kierowcy:

Hasło:

Zaloguj się

ZGŁOŚ PROBLEM

Rysunek 3-9 Projekt okna logowania do aplikacji (źródło: opracowanie własne)

Stwierdziwszy problem, kierowca może zapoczątkować proces kończący się zapisaniem i przekazaniem treści zgłoszenia. Należy wybrać opcję „zgłoś problem”, po czym ukaże się kolejne okno umożliwiające sprecyzowanie napotkanego problemu. Okno odpowiedzialne za wyświetlenie formularza ukazuje rysunek „Projekt okna zgłaszania problemu”.

Zgłaszczenie problemu

Kategoria zdarzenia

- Problem z oprogramowaniem
- Problem z logowaniem
- Problem z pojazdem
- Inny problem

Opis zdarzenia

pole opcjonalne

Zgłoś problem

Rysunek 3-10 Projekt okna zgłaszania problemu (źródło: opracowanie własne)

Formularz jest podzielony na dwie części, pierwsza, po lewej odpowiada za określenie ogólnej kategorii zgłoszanego zdarzenia poprzez wybranie go z listy kilku dostępnych. Użytkownik wybiera jedną z nich. Po prawej stronie, po wyborze kategorii istnieje możliwość dodatkowego opisania szczegółów zajścia, jeżeli wcześniej wybrana kategoria zdarzenia jest zbyt ogólna i wymaga doprecyzowania. Przyciskiem „zgłoś problem” wysyła się treść formularza od osoby odpowiedzialnej za czynności dotyczące reagowania na zdarzenia zgłasiane przez kierowców.

3.3.2. Zakładka 1: Trasa

Główne i domyślne okno w aplikacji, przedstawione w rysunku „Projekt okna "Trasa"”. Wyświetla w centralnym obszarze ekranu urządzenia mapę z wytyczoną trasą i aktualną, na bieżąco aktualizowaną, pozycją autobusu. Górną część okna aplikacja odpowiada za przekazywanie informacji kierowcy o aktualnym przebiegu trasy, wraz z naniesionymi punktami z lokalizacją przystanków na niej. W lewym górnym rogu ukazany będzie aktualny całkowity czas przejazdu autobusu, wraz z ewentualną informacją o spóźnieniu, lub czasem będącym poniżej tego ujętego w rozkładzie jazdy.

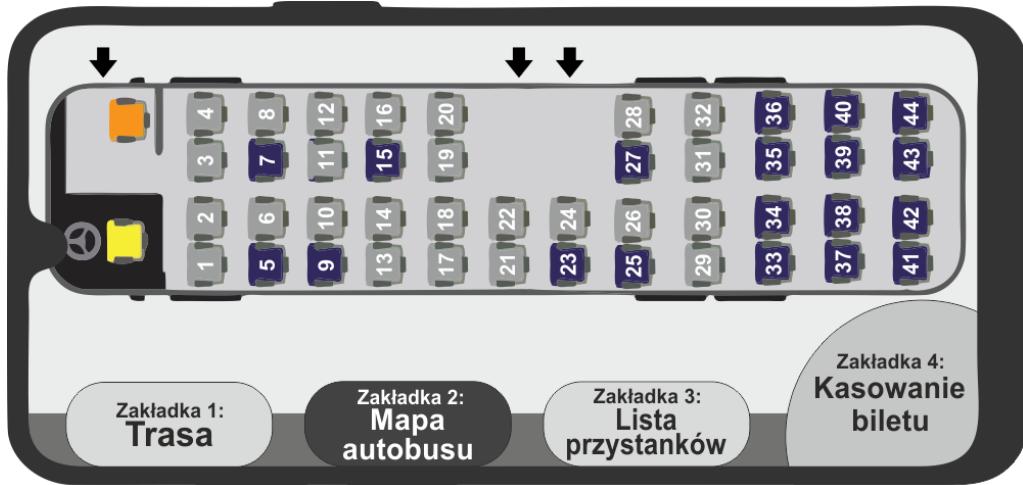


Rysunek 3-11 Projekt okna "Trasa" (źródło: opracowanie własne)

W prawym górnym rogu zaś widnieje informacja o następnym przystanku znajdującym się na trasie. Jego zmiana będzie następować automatycznie, na podstawie aktualnej pozycji autobusu na drodze. W trzeciej linijce, pod nazwą przystanku będzie podany czas przyjazdu i odjazdu z przystanku zgodny z aktualnym harmonogramem przewoźnika korzystającego z budowanego systemu informatycznego.

3.3.3. Zakładka 2: Mapa autobusu

Główna funkcjonalność, za którą odpowiada wywołanie drugiej zakładki, dostarcza mapę z realnym rozmieszczeniem miejsc w prowadzonym przez kierowcę autobusie. Kolorem szarym oznaczone będą miejsca wolne, niebieskim realnie zajęte. Zapełnienie autobusu aktualizowane jest na bieżąco, w trakcie kursu.



Rysunek 3-12 Projekt okna z mapą autobusu (źródło: opracowanie własne)

„Zajęcie” miejsca będzie następowało po skasowaniu biletu przez kierowcę. Numer siedzenia nadany na bilecie odpowiadać będzie numerowi siedzenia w autobusie. Kierowca, w ten sposób, widzi stopień zapełnienia pojazdu w czasie rzeczywistym. Ponadto, zaistnieje możliwość weryfikacji, czy zajmowane miejsca przez pasażerów pokrywają się z tymi nadanymi na bilecie. Podobnie będzie działało „zwolnienie” miejsca przez pasażera. System wykryje sytuację, w której autobus będzie zbliżał się do przystanku, w chwili jego zatrzymania automatycznie zostaną zwolnione miejsca podróżujących do danego przystanku a kierowca zostanie poinformowany przez aplikację o ilości pasażerów, którzy powinni zakończyć podróż w miejscu, w którym aktualnie znajduje się pojazd.

3.3.4. Zakładka 3: Lista przystanków

Dotknięcie przez kierowcę „Zakładki 3: Lista przystanków” ukaże aktualny spis przystanków, wraz z godzinami przyjazdu i odjazdu, będących na trasie przypisanej do aktualnego kursu. Kierowca ma możliwość interakcji z tabelą przystanków, szybko ją przeglądać czy planować, będąc wcześniej poinformowanym o problemach na drodze, objazdy do następnego punktu trasy.

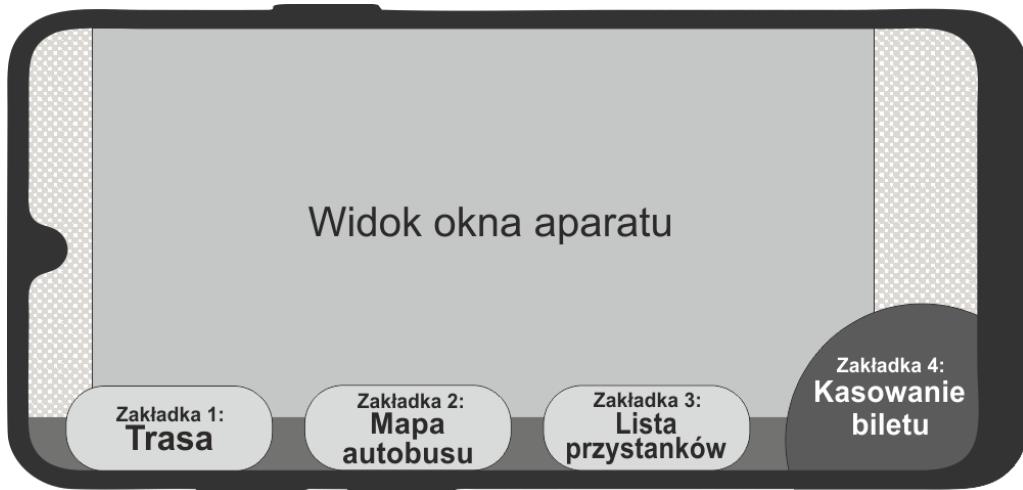


Rysunek 3-13 Projekt okna z listą przystanków (źródło: opracowanie własne)

W wyświetlnym harmonogramie zaznaczonym będzie następny przystanek, na którym ma się zatrzymać autobus, wraz z ukazaniem różnicy w czasach wynikających z sytuacji na drodze, itp. to menu aplikacji ma zadanie jedynie informacyjne.

3.3.5. Zakładka 4: Kasowanie biletu

Wchodząc w „Zakładkę 4: Kasowanie biletu” kierowca uruchamia aparat urządzenia, z którego korzysta w trakcie postoju na danym kursie. Kasowanie biletu nastąpi po zwróceniu obiektywu aparatu na kod QR biletu pasażera. Kod QR należy umieścić w ramce w oknie aparatu.



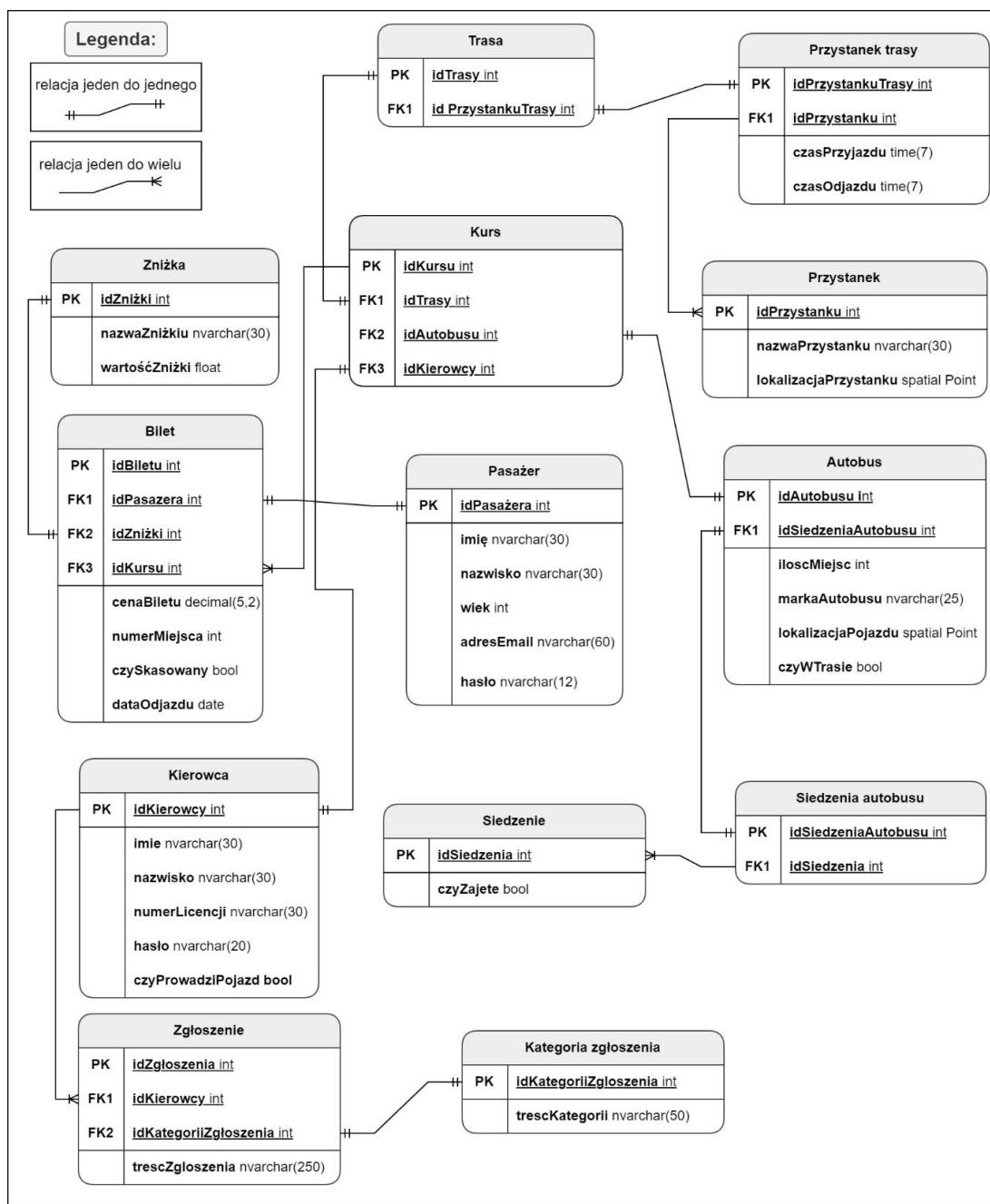
Rysunek 3-14 Projekt okna kasowania biletów (źródło: opracowanie własne)

System zaraportuje powodzenie lub niepowodzenie skasowania biletu odpowiednim komunikatem. Po jego wyświetleniu, nastąpi kolejna próba zeskanowania kodu QR, czy to

jeszcze raz tego samego, czy już nowego. Funkcjonalność ta, „zajmuje” miejsca w autobusie i odpowiednio nanosi je na wcześniej wspomnianą mapę autobusu.

3.4. Projekt bazy danych

W bazie danych przewiduje się utworzenie trzynastu tabel. Przedstawione zostały na poniższym diagramie. Każdą z nich krótko charakteryzowano poniżej. Nagłówek tabeli stanowi jej tytuł. Tuż pod nim znajduje się pole z identyfikatorem tabeli, kluczem głównym - PK. W większości przypadków, poniżej klucza głównego znajdują się klucze obce – FK. Ostatnią część tabeli stanowią pola z danymi. Po prawej stronie wytluszczonej nazwy pola w tabeli ukazane są typy danych użyte przy konstruowaniu bazy danych.



Rysunek 3-15 Projekt bazy danych (źródło: opracowanie własne)

Przy danych tekstowych, w większości przypadków, użyto typu „nvarchar”. Obsługuje on znaki inne niż pochodzące z alfabetu łacińskiego oraz Unicode, standard obsługujący większość światowych alfabetów wraz ze znakami specjalnymi. Może zawierać nie więcej niż 4000 znaków. Dane logiczne, odpowiedzialne za stan (0, 1, true, false) obsługuje typ danych „bool”. Wartości kluczy głównych i obcych przechowują pola z typem całkowitoliczbowym dodatnim „int”. W kilku miejscach występują pola z wartościami „pieniężnymi”, te przechowywane będą przez kolumny z typami dziesiętnymi „decimal”.

Baza danych budowanego systemu będzie pracować z następującymi tabelami:

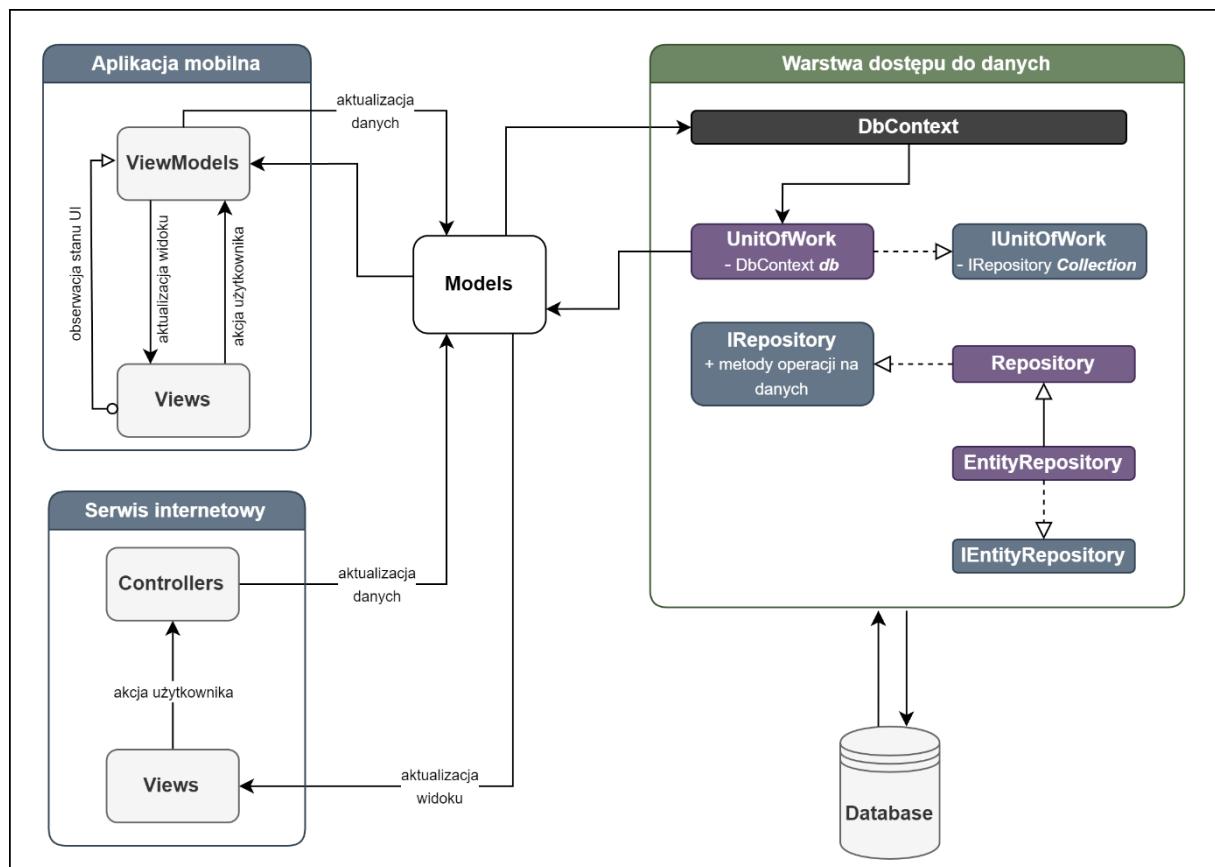
1. Kategoria zgłoszenia – tabela słownikowa przechowująca możliwe do wyboru przez kierowcę typy zgłoszeń, np.: awaria pojazdu, zdarzenie komunikacyjne.
2. Zgłoszenie – ma swoją kategorię, opcjonalną treść, numer identyfikacyjny kierowcy zgłaszającego.
3. Kierowca – przechowywane będą dane dotyczące imienia i nazwiska kierowcy, jego numeru pozwolenia na prowadzenie autobusu, oraz pole określające czy kierowca jest aktualnie w trasie.
4. Bilet – tabela mająca trzy klucze obce z tabelami: pasażer, zniżka i kurs. Każdy bilet będzie miał miejsce w autobusie wybrane przez pasażera, cenę oraz zmienną logiczną stanowiącą o tym, czy został skasowany.
5. Zniżka – tabela słownikowa z aktualnymi zniżkami. Zawiera nazwę zniżki oraz jej wartość w procentach.
6. Siedzenie – tabela przechowuje identyfikator siedzenia i wartość logiczną odpowiadającą za jego zajętość w pojeździe.
7. Siedzenia autobusu – tabela pośrednicząca w relacji wiele do wielu pomiędzy tabelami siedzenie i autobus. Ma ona swój numer id, oraz klucz obcy w postaci idSiedzenia.
8. Autobus – przede wszystkim posiada klucz obcy w postaci idSiedzeniaAutobusu, co stanowi odwołanie do zbioru siedzeń. Ponadto ma określona ilość siedzeń, markę, kolumnę odpowiadającą za dane przestrzenne, na podstawie których pojazd będący w trasie będzie lokalizowany, oraz kolumnę z wartościami logicznymi dotyczącymi stanu pojazdu: będącego w trasie, lub nie.
9. Przystanek – każdy z nich będzie posiadał swoją nazwę, oraz lokalizację w typie przestrzennym danych
10. Trasa – ciąg następujących po sobie przystanków. Tabela z relacją jeden do jednego z tabelą przystanki.

11. Przystanek trasy – tabela pośrednicząca w relacji wiele do wielu pomiędzy tabelami przystanki i trasa. Stanowi rozkład trasy z czasem przyjazdu i odjazdu autobusu z danego przystanku.
12. Pasażer – użytkownik serwisu internetowego. Tabela z kolumnami: imię, nazwisko, wiek, email, hasło. Jest w relacji jeden do jednego z tabelą „bilet”.
13. Kurs – tabela będąca w relacjach jeden do jednego z tabelami: „trasa”, „autobus”, „kierowca”. Każdy kurs odbędzie się na wcześniej wytyczonej trasie, zostanie mu przypisany wolny autobus z kierowcą.

Baza danych opierająca się na powyższym projekcie powinna zapewnić wystarczający dostęp do danych i ich przetwarzania w systemie wspomagającym krajowy transport autobusowy.

3.5. Architektura systemu

Budowany system składać się będzie z czterech komponentów: aplikacji mobilnej, serwisu internetowego i warstwy dostępu do danych, oraz bazy danych. Rysunek „Diagram architektury systemu wspomagającego krajowy transport autobusowy” przedstawia te komponenty, zależności i przepływy danych między nimi.



Rysunek 3-16 Diagram architektury systemu wspomagającego krajowy transport autobusowy. (źródło: opracowanie własne)

Czarne strzałki z pełnym grotem pokazują kierunek przepływu informacji pomiędzy komponentami. Na przykład, widoki w aplikacji mobilnej komunikują się z ViewModels w celu aktualizacji interfejsu użytkownika na podstawie interakcji użytkownika lub zmian danych, za które, z kolei, odpowiadają modele (Models).

Rozdzielenie całego systemu na pojedyncze, oddzielne i współpracujące ze sobą komponenty pozwolą na jego łatwiejszy późniejszy rozwój i implementację kilku modułów jednocześnie. W systemie przewiduje się:

1. Aplikację mobilną z dwiema powłokami:

- Views – widoki, komponenty interfejsu użytkownika, w których odbywają się interakcje użytkownika z aplikacją. „Obserwują” ViewModels pod kątem wszelkich zmian danych.
- ViewModels: „łączy” między widokami i modelami, zawierające logikę interfejsu użytkownika. Komponenty przygotowujące dane do wyświetlenia w interfejsie użytkownika i reagujące na interakcje.

2. Serwis internetowy mający dwie warstwy:

- Views - podobnie jak w przypadku aplikacji mobilnej, są to komponenty odpowiedzialne za interfejs użytkownika.
- Controllers – obsługują przychodzące żądania HTTP wynikające z działań użytkownika, manipulują modelami i wybierają widoki do wygenerowania użytkownikowi.

3. Warstwę dostępu do danych:

- DbContext – element mapowania obiektowo-relacyjnego, klasa zarządzająca kontekstem i operacjami bazy danych. Na podstawie wcześniej zdefiniowanych modeli (Models) tworzy struktury odpowiadające tabelom w bazie danych.
- UnitOfWork – klasa wzorca projektowego grupująca najczęściej wykonywane operacje. Zawiera DbContext i odpowiada za wymianę danych z bazą danych.
- IUnitOfWork – Interfejs definiujący kontrakt dla komponentu UnitOfWork, zapewniający, że każda klasa implementująca ten interfejs będzie miała określoną strukturę.
- IRepository – interfejs definiujący podstawowe operacje na danych (CRUD) wykonywane na encjach.
- Repository – klasa implementująca IRepository, zawiera rzeczywiste metody operacji na danych.

- EntityRepository – określony typ repozytorium, który działa na określonym typie encji.
 - IEntityRepository – interfejs implementowany przez EntityRepository, określający operacje dostępne dla danej encji, wraz ze specyficznymi tylko dla tej encji metodami, nigdzie wcześniej nie wykorzystywanyimi.
4. Modele – struktura danych w aplikacji. Aktualizowane przez komponenty ViewModel i Controller.
 5. Bazę danych

Architektura przedstawiona powyżej wyraźnie oddziela zadania systemu ułatwiając zarządzanie nim, konserwację i jego skalowanie. Pozwala na to wykorzystanie wzorców repozytorium (ang.: repository) i jednostka pracy (ang.: unit of work). Jest to powszechnie podejście do tworzenia skalowalnych i łatwych w utrzymaniu aplikacji. [12] [16]

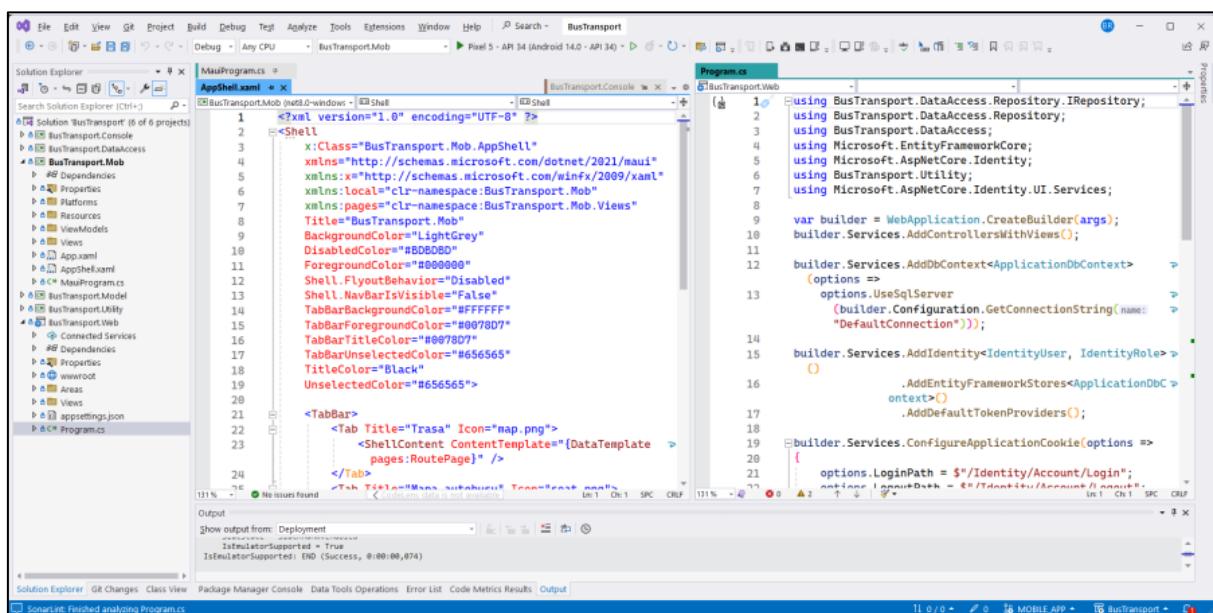
4. IMPLEMENTACJA

Końcowy rozdział pracy inżynierskiej zostanie poświęcony fazie implementacji wcześniej projektowanego systemu informatycznego. Pierwszym jego etapem będzie utworzenie bazy danych gotowej do pracy z pozostałymi modułami całego systemu.

Następnie zaimplementowana zostanie aplikacja przeglądarkowa – serwis dla pasażera. Omówiony zostanie proces tworzenia interfejsu użytkownika wraz z implementowaną funkcjonalnością.

Drugi, duży implementowany moduł to aplikacja mobilna dla kierowcy budowana we frameworku .NET MAUI. Aplikacja będzie powstawała jedynie na system Android. Całość oparta zostanie o wzorzec MVVM - Model-View-ViewModel, który szczegółowo zostanie przedstawiony w odpowiednim podrozdziale.

System będzie budowany w środowisku programistycznym Visual Studio 2022. Zostało założone w nim rozwiązanie (ang.: „solution”), „BusTransport”. Rozwiązaniem zaś nazywamy kontener na mniejsze składniki wchodzące w całąację solucję. Składniki te Visual Studio nazywa projektami. Odpowiadają one już bezpośrednio za warstwy i funkcjonalność aplikacji, na przykład: projekty dotyczące dostępu do danych czy interfejsu użytkownika.

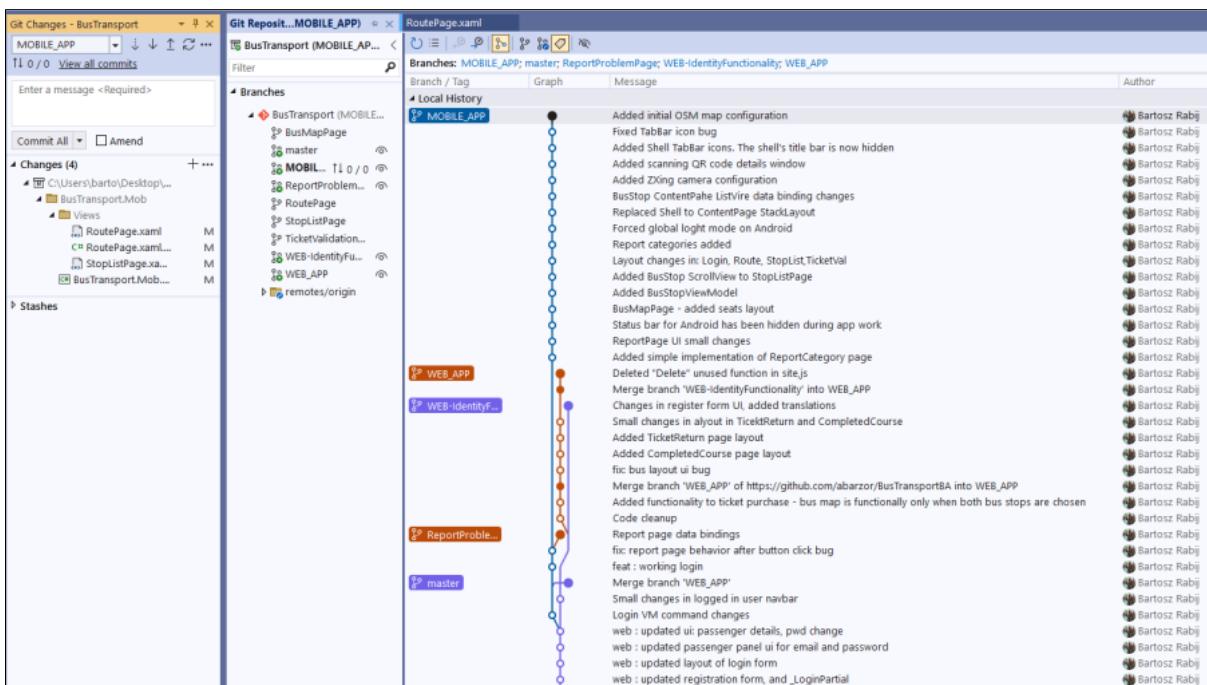


Rysunek 4-1 Widok środowiska programistycznego dla rozwiązania "BusTransport" (źródło: opracowanie własne)

Obrazek „Widok środowiska programistycznego dla rozwiązania "BusTransport"” pokazuje wszystkie projekty, części budowanego oprogramowania. Po lewej stronie okna widnieje moduł odpowiedzialny za wyświetlanie eksploratora całego rozwiązania („Solution Explorer”), oraz zakładka „Git Changes” pozwalająca na organizację kodu w repozytorium. Na rozwiązanie „BusTransport” składają się poszczególne projekty: „Mobile” odpowiadający za

aplikację dla kierowcy, oraz „Web” implementujący serwis internetowy dla pasażera. Za modelowanie danych odpowiada „Models”. To w nim określone są wszystkie encje, klasy napisane w języku C#, odpowiadające strukturom w bazie danych. Są one następnie w „DataAccess” mapowane na tabele i zapisywane w bazie danych, do której zostało wcześniej skonfigurowane połączenie.

Pracując z kodem należy śledzić jego zmiany w czasie. Wiele razy można napotkać się z sytuacją potencjalnie bez wyjścia, kiedy to zajdzie potrzeba powrócenia do wersji kodu sprzed dawna. Istnieją specjalne rozwiązania służące do śledzenia zmian w kodzie, zwane systemami kontroli wersji. Podczas tworzenia systemu wspomagającego pracę transportu autobusowego wykorzystany system kontroli wersji Git. Zapewnia on efektywną współpracę, zarządzanie kodem i bezpieczeństwo projektu. Główną jego cechą jest śledzenie i rejestrowanie zmian w kodzie, co pozwala na łatwe wycofywanie lub modyfikowanie poszczególnych fragmentów kodu bez ryzyka utraty dotychczasowych wyników. Git wspiera pracę zespołową, umożliwiając wielu twórcom oprogramowania równoczesny rozwój różnych jego części bez konfliktów z innymi programistami pracującymi nad tym samym rozwiązaniem. Opiera się on na gałęziach (ang. „branch”) – niezależne linie rozwoju w projekcie pracę nad różnymi funkcjami, eksperymentami lub poprawkami, bez wpływu na główną linię kodu, zwaną często gałęzią "master" lub "main". Każdą z nich można scalić z główną gałęzią projektu. [12]



Rysunek 4-2 Okno odpowiadające za zarządzanie repozytorium Gita w Visual Studio (źródło: opracowanie własne)

Zmiany wprowadzone w kodzie rejestrowane są za pomocą mechanizmu „commit”. Każdy z nich stanowi zatwierdzenie zmian na danej gałęzi, wraz z opisem tych zmian i jest

wysłany do repozytorium kodu. Do każdego „commita” można wrócić, przywracając zaktualizowaną nim wersję kodu. Rysunek „Okno odpowiadające za zarządzanie repozytorium Gita w Visual Studio” pokazuje interfejs graficzny jednego ze zintegrowanych elementów środowiska Visual Studio graficznie reprezentujący zmiany w repozytorium. W centralnej części ekranu znajduje się graf reprezentujący historię zmian, „commitów” w repozytorium wraz z danymi autora, które je wykonał. W tym przypadku pokazywane jest pięć gałęzi: „master”, „MOBILE_APP”, „ReportProblemPage”, „WEB-identityFunctionality” czy „WEB_APP”. Każda z nich dotyczy odrębnej funkcjonalności w systemie. Po lewej stronie ekranu widoczna jest lista plików, w których nastąpiły zmiany i są one gotowe do zatwierdzenia poprzez „commit”. Po prawej stronie nazwy pliku widnieje litera M – z języka angielskiego - "Modified", oznacza, że kod w pliku został zmodyfikowany względem poprzedniego „committa”. W przypadku usunięcia pliku, jego nazwa zostanie również uwzględniona na liście zmian, będzie oznaczony literą „D” (ang.: delete), a jego nazwa zostanie przekreślona. Pliki dodawane do repozytorium oznaczone będą literą statusu „A” (ang.: added).

W centrum okna i po prawej jego stronie widoczne są detale dotyczące historii zmian. Każdy wiersz reprezentuje inny „commit” z krótkim opisem zmian, które zostały nim wprowadzone.

4.1. Implementacja bazy danych

By oprogramowanie mogło skutecznie i szybko pozyskiwać dane, pracować na nich i edytować je potrzebny jest wydajny silnik bazodanowy. Od początkowych koncepcji systemu wspierającego krajowy transport autobusowy rozważany był jeden typ bazy danych, relacyjny, czyli taki, który przedstawia dane w tabelach wraz ze zdefiniowaniem logicznego połączenia między nimi.

4.1.1. Omówienie sposobu implementacji bazy danych metodą „code first”

Do przygotowania tabel w budowanej bazie danych wykorzystano jeden ze sposobów i podejść do tworzenia struktur bazodanowych - ORM, object-relational mapping, mapowanie obiektowo-relacyjne. Podejście takie jest najbardziej odpowiednie dla budowanego systemu. System ten wykorzystuje inne mechanizmy obiektowe, sam język programowania – C# jest w pełni obiektowy, inne jego komponenty stanowią współpracujące ze sobą obiekty. Sama technika ORM tworzy pewien „pomost” pomiędzy programem a, w większości przypadków, relacyjną bazą danych. ORM jest postrzegane jako warstwa łącząca programowanie zorientowane obiektowo ze strukturą bazodanową poprzez wykorzystanie języka

programowania, co upraszcza komunikację z bazą danych podczas tworzenia oprogramowania. Nie używany jest język typowy dla zapytań bazodanowych, taki jak SQL, tylko, dzięki narzędziom ORM, istnieje możliwość wykonywania operacji na danych wcześniej przygotowanymi i wkomponowanymi w obiektowy język programowania, funkcjami odpowiadającymi zapytaniom pisany bezpośrednio w SQL.

Praktycznie każdy współczesny obiektowy język programowania ma swoje dedykowane narzędzia ORM. Platforma .Net i jej główny język programowania, C# korzysta z rozwiązania „Entity Framework”.

Tenże framework jest częścią ADO.NET, czyli zestawu technologii ujętych w platformie .NET, które umożliwiają dostęp do źródeł danych, takich jak bazy danych, pliki XML czy usługi internetowe. ADO.NET zapewnia mechanizmy do pobierania, aktualizacji, wstawiania i usuwania danych w różnych źródłach danych. Umożliwia implementację bazy danych w różnych podejściach:

- „Database first” – by móc zacząć pracować z danymi w kodzie powstającej aplikacji potrzebna będzie wcześniej przygotowana baza danych. Na jej podstawie powstaje model danych wykorzystywany w kodzie umożliwiający wymianę danych pomiędzy nimi. Na podstawie tabel z bazy danych Entity Framework generuje klasy reprezentujące strukturę danych w bazie w formie klas, obiektów w kodzie im odpowiadających. W toku procesu powstaje plik .edmx, zawierający mapowanie między bazą danych a modelem danych. W kodzie odwoływać się można do nich tak jak do innych obiektów
- „Code first” – model danych definiowany jest za pomocą kodu. Następnie w oparciu o ten model generowana jest baza danych. Pierwszym krokiem tego podejścia będzie zdefiniowanie klas encji w taki sposób, by te odpowiadały tabelom w budowanej bazie danych. Następnie używany jest mechanizm migracji, by gotowe tabele urzeczywistnić w bazie danych. Podejście to jest szczegółowo opisane w kolejnym podrozdziale.

Podczas tworzenia systemu dla transportu autobusowego wykorzystywano metodę „code first”.

4.1.2. Utworzenie i mapowanie klas jako gotowych tabel w bazie danych

W budowanym systemie zostały utworzone klasy będące odpowiednikiem tabel w bazie danych: Bus, BusStop, Course, Discount, Driver, Passenger, Report, ReportCategory, Route,

Seat, Ticket, SeatsOfBus, RouteStop. Każda z nich posiada odrębne „właściwości” (ang.: „properties”), odpowiadające za osobną kolumnę w tabeli.

```
public class BusStop
{
    23 references
    public int BusStop_Id { get; set; }
    18 references
    public string StopName { get; set; }
    16 references
    public double BusStopLocationX { get; set; }
    16 references
    public double BusStopLocationY { get; set; }
}

public class Bus
{
    1 reference
    public int Bus_Id { get; set; }
    0 references
    public int NumberOfSeats { get; set; }
    0 references
    public string VehicleBrand { get; set; }
    0 references
    public bool IsOnRoad { get; set; } = false;
    0 references
    public double BusLocationX { get; set; }
    0 references
    public double BusLocationY { get; set; }

    0 references
    public Course Course { get; set; }
}
```

Rysunek 4-3 Klasy Bus i BusStop budowanego modelu danych (źródło: opracowanie własne)

Dla przykładu: dwie powyższe klasy, BusStop i Bus, reprezentują w bazie danych tabele odpowiadające za przechowywanie danych dotyczące przystanków autobusowych i autobusów. Pierwsza z nich posiadać będzie kolumny z unikalnym identyfikatorem, nazwą przystanku i jego lokalizacją na mapie. Druga, identyfikator, ilość siedzeń, markę pojazdu, położenie w formie współrzędnych geograficznych, oraz właściwość Course, dzięki której utworzona zostanie relacja jeden do jednego z tabelą Course, to tzw. „właściwość nawigacyjna” (ang.: „navigation property”) odpowiadającą za utworzenie klucza obcego i, w tym przypadku, relacji jeden do jednego ze wcześniej wspomnianą tabelą.

Mając przygotowane klasy należy określić, która jej właściwość będzie odpowiadała za klucz główny, unikalny identyfikator dla każdego rekordu w tabeli bazy danych. Można też precyzyjnie określić ograniczenia, na przykład długość danych, czy ilość cyfr po przecinku. Bardzo pomocne przy tym jest „Fluent API” będące zorientowaną obiektywną metodą konfiguracji modelu danych za pomocą „łańcucha metod”, łączącego kilka metod następujących jedna po drugiej. Takie podejście oferuje bardziej rozbudowane i elastyczne opcje konfiguracyjne niż atrybuty, umieszczane przy właściwościach w klasach modelu danych.

Zaczynając projektowanie bazy danych należy pobrać i zainstalować, za pomocą wbudowanego w Visual Studio 2022 narzędzia – menedżera pakietów NuGet, kilka pakietów umożliwiających takie podejście:

- Microsoft.EntityFrameworkCore – kluczowy składnik Entity Framework, zapewnia jego najważniejszą funkcjonalność. Zawiera klasę DbContext którą musi dziedziczyć

stworzona przez programistę klasa mająca definiować model bazy danych. Umożliwia wykorzystywanie LINQ (Language Integrated Query) służące do wykonywania zapytań do bazy danych przy użyciu składni języka C#. Ponadto pakiet ten umożliwia korzystanie z mechanizmu rozwijania bazy danych - migracji oraz konfiguracji encji..

- Microsoft.EntityFrameworkCore.SqlServer – pakiet rozszerzeń zapewniający obsługę SQL Server jako serwera bazodanowego. Jego komponenty tłumaczą polecenie i zapytania napisane w C# na składnię specyficzną dla SQL Server. Obsługiwane też są typy danych specyficzne dla SQL Server, na przykład: smalldatetime, nvarchar, geography, geometry.
- Microsoft.EntityFrameworkCore.Tools - zestaw narzędzi wiersza poleceń szczególnie tych, związanych z migracjami.

Przygotowane wyżej wymienione narzędzia pozwalają, by przystąpić do stworzenia klas reprezentujących obiekty w bazie danych. Każda klasa odpowiada jednej tabeli w bazie danych, a właściwości klasy odpowiadają kolumnom. Na podstawie trzech klas: Passenger, Ticket oraz Discount pokazanych na rysunku zostanie przedstawiony proces konfiguracji kluczy głównych, kluczy obcych i ograniczeń pól w przyszłych tabelach w bazie danych.

```

public class Passenger
{
    7 references
    public int Passenger_Id { get; set; }
    5 references
    public string Name { get; set; }
    5 references
    public string Surname { get; set; }
    4 references
    public int Age { get; set; }
    5 references
    public string Email { get; set; }
    4 references
    public string Password { get; set; }

    1 reference
    public Ticket Ticket { get; set; }
}

public class Ticket
{
    6 references
    public int Ticket_Id { get; set; }
    4 references
    public string StartStop { get; set; }
    4 references
    public string EndStop { get; set; }
    4 references
    public decimal TicketPrice { get; set; }
    5 references
    public int SeatNumber { get; set; }
    5 references
    public DateTime DepartureDate { get; set; }
    3 references
    public bool IsVaild { get; set; } = false;

    3 references
    public int Passenger_Id { get; set; }
    1 reference
    public Passenger Passenger { get; set; }

    5 references
    public int Discount_Id { get; set; }
    1 reference
    public Discount Discount { get; set; }
}

```

```

public class Discount
{
    8 references
    public int Discount_Id { get; set; }
    9 references
    public string DiscountName { get; set; }
    6 references
    public float DiscountValue { get; set; }

    1 reference
    public Ticket Ticket { get; set; }
}

```

Rysunek 4-4 Trzy klasy w C# przygotowane do mapowania bazy danych (źródło: opracowanie własne)

Po przygotowaniu klas, należy przygotować kolejną, reprezentującą kontekst bazy danych. Poniższy rysunek przedstawia utworzoną wcześniej klasę kontekstu ApplicationDbContext dziedziczącą po klasie DbContext.

```

17 references
6   □
7   ►
8     0 references
9       public DbSet<Bus> Buses { get; set; }
10      1 reference
11        public DbSet<BusStop> BusStops { get; set; }
12        0 references
13          public DbSet<Course> Courses { get; set; }
14          0 references
15            public DbSet<Driver> Drivers { get; set; }
16            1 reference
17              public DbSet<Passenger> Passengers { get; set; }
18              0 references
19                public DbSet<Route> Routes { get; set; }
19                  1 reference
20                    public DbSet<Ticket> Tickets { get; set; }
21                    0 references
22                      public DbSet<Seat> Seats { get; set; }
23                        0 references
24                          public DbSet<Report> Reports { get; set; }
25                          0 references
26                            public DbSet<ReportCategory> ReportCategories { get; set; }
27                            1 reference
28                              public DbSet<Discount> Discounts { get; set; }
29                              0 references
30                                public DbSet<SeatsOfBus> SeatsOfBuses { get; set; }
31                                  0 references
32                                    public DbSet<RouteStop> RouteStoppes { get; set; }

```

Rysunek 4-5 Właściwości typu DbSet mapowane na tabele w bazie danych. (źródło: opracowanie własne)

DbContext zapewnia interakcję z bazą danych. Jest pomostem między nią a logiką biznesową. Dzięki niej jest możliwe otwieranie i zamykanie połączeń w zależności od potrzeb, a także zarządzanie transakcjami. Pozwala na korzystanie z właściwości DbSet<T>.

Każda z wcześniejszej przygotowanych klas, która ma zostać mapowana na bazę danych w „ApplicationDbContext” ma swój DbSet<T> - typ reprezentujący kolekcję określonego typu <T>, gdzie T odpowiada wcześniej utworzonej klasie.

```

37   □
38   ►
39     protected override void OnModelCreating(ModelBuilder modelBuilder)
40     {
41       modelBuilder.Entity<Ticket>().HasKey(t => t.Ticket_Id);
42
43       modelBuilder.Entity<Ticket>().HasOne(p => p.Passenger)
44         .WithOne(p => p.Ticket)
45           .HasForeignKey<Ticket>(p => p.Passenger_Id);
46       modelBuilder.Entity<Discount>().HasOne(p => p.Ticket)
47         .WithOne(p => p.Discount)
48           .HasForeignKey<Ticket>(p => p.Discount_Id);
49
50       modelBuilder.Entity<Ticket>().Property(t => t.StartStop).IsRequired();
51       modelBuilder.Entity<Ticket>().Property(t => t.StartStop).HasMaxLength(maxLength: 50);
52
53       modelBuilder.Entity<Ticket>().Property(t => t.EndStop).IsRequired();
54       modelBuilder.Entity<Ticket>().Property(t => t.EndStop).HasMaxLength(maxLength: 50);
55
56       modelBuilder.Entity<Ticket>().Property(t => t.TicketPrice).IsRequired();
57       modelBuilder.Entity<Ticket>().Property(t => t.TicketPrice).HasPrecision(precision: 5, scale: 2);
58
59       modelBuilder.Entity<Ticket>().Property(t => t.SeatNumber).IsRequired();
60
61       modelBuilder.Entity<Ticket>().Property(t => t.DepartureDate).IsRequired();
62
63       modelBuilder.Entity<Ticket>().Property(t => t.IsVaild).IsRequired();

```

Rysunek 4-6 Łańcuchy metod konfigurujące klucze i kolumny w bazie danych. (źródło: opracowanie własne)

Kolejnym krokiem w projektowaniu bazy danych w podejściu code-first jest konfiguracja tabel. OnModelCreating jest metodą klasy ApplicationContext, dostarczającą precyzyjną kontrolę nad sposobem mapowania klas na bazę danych. W jej ciebie, poprzez używanie parametru modelBuilder określone będą wszystkie ograniczenia kolumn w tabelach. Rysunek powyżej przedstawia taką konfigurację dla tabeli odpowiedzialnej za przechowywanie danych biletów. W linijce 39 kodu na powyższym rysunku, używając podejścia Fluent API, określono klucz główny:

- `.Entity<Ticket>()` - najpierw określono encję, jaka ma być konfigurowana
- `.HasKey(t => t.Ticket_Id);` - potem, wykorzystując wyrażenie lambda, określa się właściwość „Ticket_Id” klasy „Ticket” jako klucz główny tabeli.

W podobny sposób można konfigurować też klucze obce. Jedna z klas przedstawionych w rysunku „Trzy klasy w C# przygotowane do mapowania bazy danych.”, klasa „Ticket” ma przygotowane dwie dodatkowe właściwości: „Passenger_ID” oraz „Ticket_Id”. Nad każdą z nich znajdują się pola reprezentujące powiązania odpowiednio ze zmiennymi kierunkowymi klas z którymi będzie tworzony będzie klucz obcy: Ticket – Passenger, oraz Ticket – Discount. Oba klucze główne zostały skonfigurowane w taki sam sposób:

- `modelBuilder.Entity<Ticket>()` – rozpoczęcie konfiguracji dla encji „Ticket”
- `.HasOne(p => p.Passenger)` – określa, że encja „Ticket” posiada relację jeden do jednego z encją „Passenger”, czyli każdy bilet jest powiązany z jednym pasażerem.
- `.WithOne(p => p.Ticket)` – encja „Passenger” również posiada relację jeden do jednego z encją „Ticket”. To oznacza, że każdy pasażer jest powiązany z jednym biletem.
- `.HasForeignKey<Ticket>(p => p.Passenger_Id);` – ustalenie lokalizacji klucza obcego używanego do mapowania tej relacji w encji „Ticket”, gdzie jest reprezentowany przez pole „Passenger_Id”. To pole właśnie, w encji „Ticket” jest kluczem obcym wskazującym na klucz główny encji „Passenger”.

Na praktycznie każdą właściwość klas encji można, w toku konfiguracji, nakładać ograniczenia. Łańcuch znaków można ograniczyć do żądanej długości, typy liczbowe uprecyzyniać co do ilości miejsc po przecinku:

- Tak samo, jak przy określaniu klucza obcego, na samym początku należy określić konfigurowaną encję.
- `.Property(t => t.TicketPrice)` – określa konkretną właściwość encji, która ma być konfigurowana. W tym przypadku jest to „TicketPrice”.

- Następnie wybraną właściwość encji możemy:
 - `.HasPrecision(2, 2);` – doprecyzować co do ilości miejsc przed przecinkiem i po nim. W tym przypadku są do dwa miejsca przed i dwa miejsca po,
 - `.IsRequired();` – określić czy ma być wymagana, co oznacza, że nie może mieć wartości null,
 - `.HasMaxLength(50);` – ograniczyć co do długości maksymalnej tekstu.
- `.Entity<Ticket>().Ignore(t => t.SeatNumber);` – właściwość "SeatNumber" encji „Ticket” zostanie pominięta przy mapowaniu i nie będzie widniała jako kolumna w bazie danych.

Skonfigurowawszy encje i ich właściwości należy przygotować połączenie z bazą danych. Visual Studio 2022 przy tworzeniu aplikacji przeglądarkowej generuje dwa kluczowe przy tym pliki:

- appsettings.json – plik konfiguracyjny przechowujący ustawienia aplikacji takie jak: parametry połączenia z bazą danych,
- Program.cs – zawiera kod startowy aplikacji. Za pomocą wstrzykiwania zależności rejestrowane są w nim kolejne, używane podczas budowania aplikacji komponenty.

Plik „appsettings.json” należy rozszerzyć o część „ConnectionStrings” w którym określmy jedno, lub więcej połączeń z bazą danych. Należy zacząć jego nazwy, w tym przypadku jest to „DefaultConnection”, gdzie po dwukropku podany jest ciąg znaków (ang.: connection string) zawierający parametry wymagane do współpracy budowanej aplikacji się z bazą danych.



```

2 {
3   "Logging": {
4     "LogLevel": {
5       "Default": "Information",
6       "Microsoft.AspNetCore": "Warning"
7     }
8   },
9   "AllowedHosts": "*",
10  "ConnectionStrings": {
11    "DefaultConnection": "Server=***;Database=BusTransport;Trusted_Connection=True;TrustServerCertificate=True"
12  }
13 }
14 }
15

```

Rysunek 4-7 Plik appsettings.json z konfiguracją połączenia z bazą danych (źródło: opracowanie własne)

Na serwerze, do którego mamy dostęp, tworzymy bazę danych o nazwie „BusTransport”, następnie, kierując się wymaganiami i potrzebami istnieje możliwość

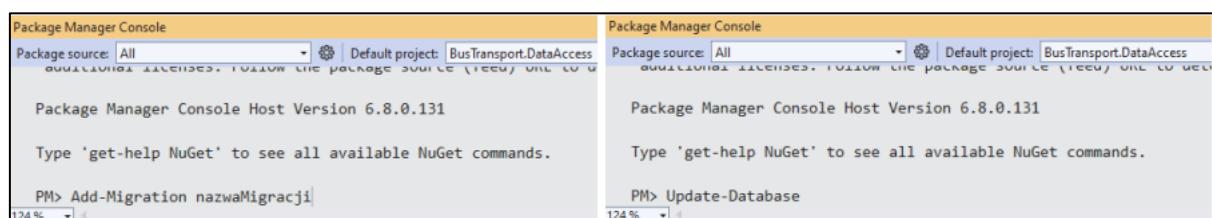
doprecyzowania innych parametrów. Możliwe jest skonfigurowanie kilku połączeń z bazą, muszą się one jedyne różnić nazwą.

```
8     builder.Services.AddDbContext<ApplicationContext>(options =>
9         options.UseSqlServer(builder.Configuration.GetConnectionString(name: "DefaultConnection")));
10    builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
11
12    var app = builder.Build();
13
14
15
```

Rysunek 4-8 Usługi dodane w pliku Program.cs (źródło: opracowanie własne)

W pliku Program.cs dodawane są usługi i metody ułatwiające konfigurowanie aplikacji. Aktualnie dodane są dwie „UnitOfWork”, jeden z używanych wzorców i podejść podczas budowania serwisu internetowego, oraz kontekst bazy danych (linijka 9 i 10), co pokazuje powyższy rysunek: „Usługi dodane w pliku Program.cs”. Metoda AddDbContext rejestruje kontekst bazy danych, w tym przypadku wcześniej utworzoną klasę ApplicationContext, poprzez wstrzykiwanie zależności (ang.: dependency injection), technikę, w której zależności klasy są wstrzykiwane z zewnątrz, a nie tworzone w samej klasie. Dalej konfigurowany jest sposób połączenia z bazą danych SQL Server. W tym przypadku używany dostawca bazy danych to Microsoft SQL Server, a parametr "DefaultConnection" wskazuje na dane dotyczące konfiguracji w pliku appsettings.json, który zawiera parametry połączeniowe (connection string) do bazy danych.

Skończona konfiguracja dostępu aplikacji do bazy danych umożliwia komunikację z nią. Platforma .NET z zainstalowanymi pakietami Entity Framework posiada wbudowany mechanizm migracji. Pomaga on zarządzać ewolucją i strukturą bazy danych wraz z rozwojem aplikacji.



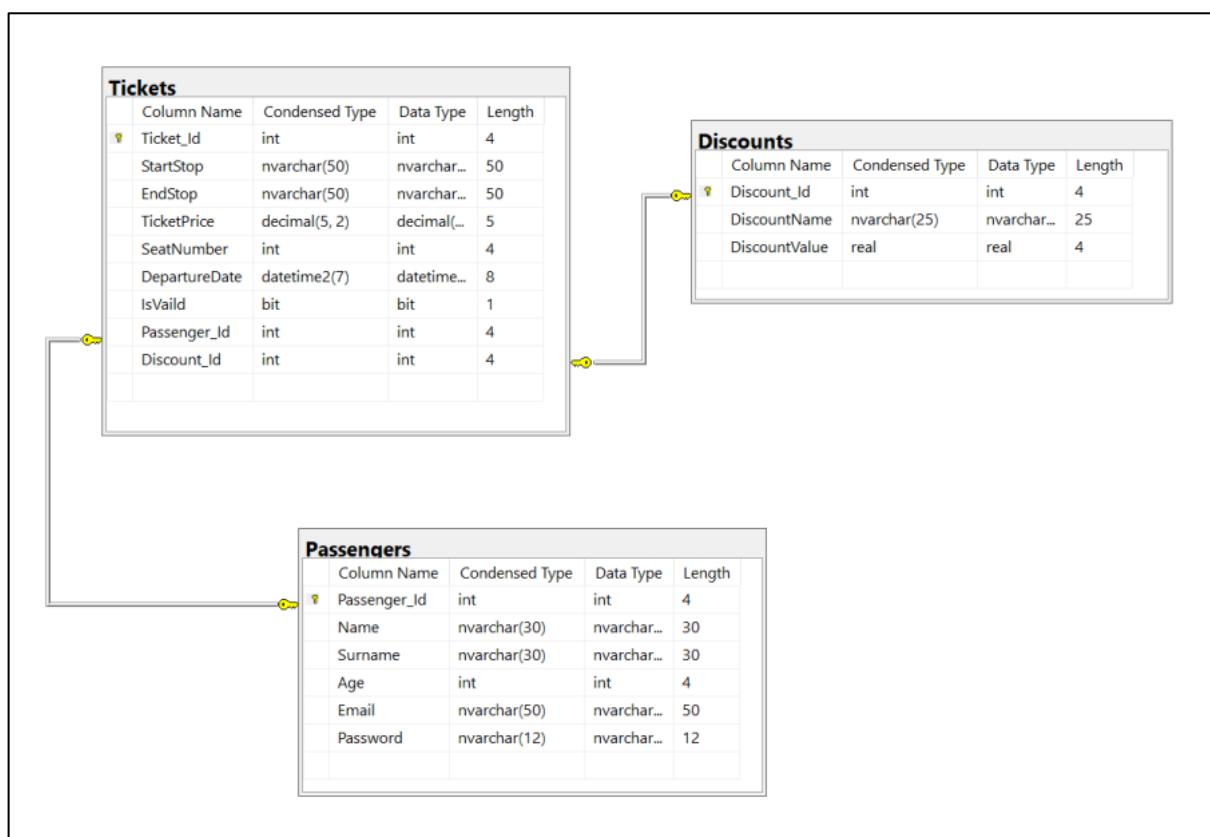
Rysunek 4-9 Okno poleceń konsoli menedżera pakietów (źródło: opracowanie własne)

Migracje skupiają się na dwóch głównych elementach:

- plikach z kodem migracji – generowane są automatycznie na podstawie zmian w klasie kontekstu bazy danych
- plikach kontekstu bazy danych – na ich podstawie mapowane są tabele w strukturze bazodanowej

W tym momencie w oknie konsole menedżera pakietów można, poleciem Add-Migration i nazwą migracji, zlecić narzędziu utworzenie plików migracji. W domyślnym miejscu, tam gdzie znajduje się klasa kontekstu bazy danych utworzy się folder „Migrations”. To on będzie przechowywał pliki wszystkich wykonanych migracji. W budowanym systemie znajduje się w osobnym projekcie „BusTransport.DataAccess”. By wcześniej przygotowane tabele mogły zaistnieć w rzeczywistej bazie danych wymagane jest, również w oknie poleceń konsoli menedżera pakietów, użycie polecenia „Update-Database”, wykonuje ono aktualizację bazy danych zgodnie z najnowszą migracją. Jeśli baza danych jest aktualna, nie wprowadzi żadnych zmian.

Migracje pozwalają na elastyczny rozwój struktury bazy danych wraz z ewolucją aplikacji. Programiści mogą łatwo dodawać, usuwać i modyfikować elementy bazy danych bez potrzeby ręcznego zarządzania skryptami SQL i zarządzać ewolucję struktury bazy danych wraz z rozwojem aplikacji.



Rysunek 4-10 Diagram gotowych tabel w bazie danych (źródło: opracowanie własne)

Rezultatem polecenia „Update-Database” w tym przypadku będzie utworzenie bazy danych o nazwie „BusTransport” a w niej trzech tabel: Tickets, Passengers, Discounts. W narzędziach służących do zarządzania można szczegółowo podejrzeć szczegółowe utworzonych tabel. Ukaże je rysunek „Diagram gotowych tabel w bazie danych”. Utworzony został w

narzędziu Microsoft SQL Server Management Studio za pomocą wbudowanej funkcjonalności tworzenia diagramów i edytowania pól jakie mają wyświetlać. [17]

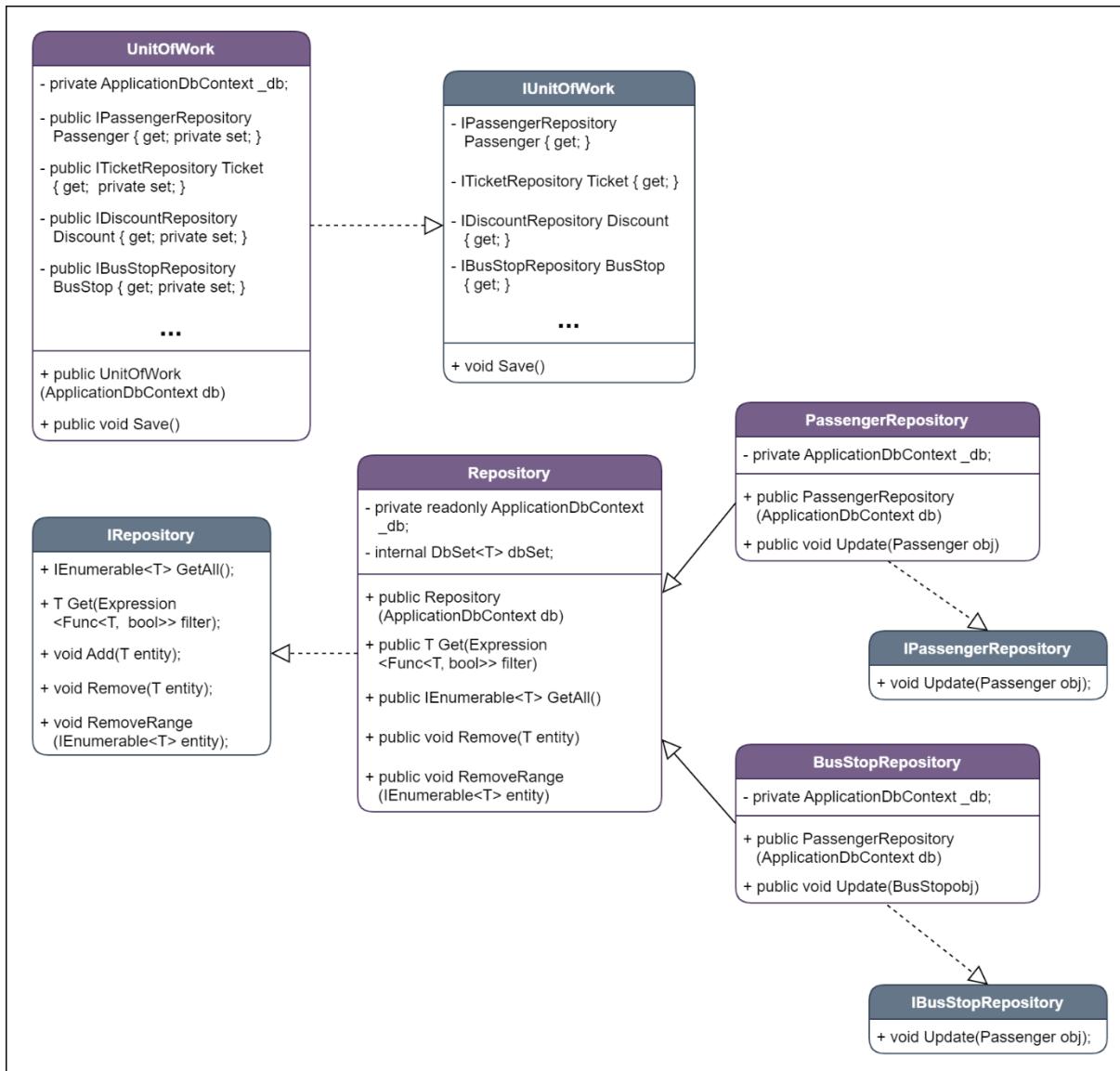
4.2. Implementacja serwisu internetowego

Jedną z dwóch części systemu wspomagającego pracę krajowego transportu autobusowego będzie serwis internetowy dla pasażera. Zaoferuje on funkcjonalność przeglądania aktualnych informacji na temat przewoźnika. Klient uzyska dostęp do swojego indywidualnego panelu z edytowalnymi danymi, podanymi podczas wcześniejszej rejestracji, i do innych funkcjonalności. W rozdziale o implementacji serwisu internetowego zostaną ukazane wykorzystane podejścia w jego tworzeniu, oraz zaprezentowane widoki zbudowanych stron w postaci zrzutów ekranu z przeglądarki internetowej. Wykorzystane wzorce projektowe zostaną omówione na podstawie napisanego kodu kontrolującego przepływ danych i reakcję serwisu na nie.

Interfejs graficzny został wykonany przy pomocy biblioteki Bootstrap. Poszczególne elementy w oknach osadzono w komponencie „card”. To elastyczny i rozszerzalny kontener na zawartość. Posiada opcje do konfigurowania nagłówków i treści zawartej w ciele. Tytuł strony, lub funkcjonalność za którą odpowiada będzie umieszczona w nagłówku komponentu „card” na szarym, lekko gradientowym polu. Treść wraz z kontrolkami znajdują się poniżej, na białym tle. Tło całej strony, pod komponentami i górna belka nawigacyjna stanowi kolor bezowy przeplatający się z odcieniami zieleni.

4.2.1. Wykorzystane wzorce i podejścia do stworzenia aplikacji przeglądarkowej

Tworząc serwis internetowy, do interakcji z danymi, wykorzystano dwa wzorce: „repozytorium” (ang.: repository) oraz „jednostka pracy” (ang.: unit of work). Pierwszy z wymienionych izoluje warstwę danych od reszty aplikacji. To wzorzec projektowy ułatwiający zarządzanie operacjami na bazie danych poprzez agregowanie wszystkich repozytoriów pod jedną wspólną abstrakcją, co ułatwia zarządzanie transakcjami i operacjami na danych. Drugi koordynuje operacje dostępu do danych i stosowany jest w kontekście interakcji z bazami danych. Oba z nich są często wykorzystywane razem do operacji na danych: tworzenia, odczytywania, aktualniania i usuwania (CRUD).



Rysunek 4-11 Diagram klas odpowiadających za warstwę dostępu do danych (źródło: opracowanie własne)

Na powyższym diagramie przedstawiono strukturę wykorzystywanych wzorców. Elementy odpowiadające klasom mają górną sekcję koloru fioletowego, a odpowiadające interfejsom, szarą. Zgodnie ze standardami UML znak pola w klasie to minus, znak metody to plus. Punktem wyjścia jest ogólny interfejs „**IRepository**”. Może być zaimplementowany dla różnych typów obiektów „**T**”. Parametr „**where T : class**” określa, że typ „**T**” musi być klasą, a nie, na przykład, typem prostym (int, char itp.). Składa się on z metod:

- ***IEnumerable<T> GetAll();*** - metoda zwracająca wszystkie obiekty typu „**T**” przechowywane w repozytorium.
- ***T Get(Expression<Func<T, bool>> filter);*** - pobiera obiektu typu „**T**” na podstawie określonego filtra. Filtr jest reprezentowany przez wyrażenie lambda, które określa warunki, które musi spełniać zwracany obiekt,
- ***void Add(T entity);*** - dodaje nowy obiekt typu „**T**” do repozytorium,

- `void Remove(T entity);` - usuwa określony obiekt typu „T” z repozytorium,
- `void RemoveRange(IEnumerable<T> entity);` - usuwa z repozytorium określony zakres obiektów przekazanych jako kolekcja typu „`IEnumerable<T>`”.

```

public class Repository<T> : IRepository<T> where T : class
{
    private readonly ApplicationDbContext _db;
    internal DbSet<T> dbSet;
    4 references
    public Repository(ApplicationDbContext db)
    {
        _db = db;
        this.dbSet = _db.Set<T>();
    }
    4 references
    public void Add(T entity)
    {
        dbSet.Add(entity);
    }
    7 references
    public T Get(Expression<Func<T, bool>> filter)
    {
        IQueryable<T> query = dbSet;
        query = query.Where(filter);
        return query.FirstOrDefault();
    }
    7 references
    public IEnumerable<T> GetAll()
    {
        IQueryable<T> query = dbSet;
        return query.ToList();
    }
    3 references
    public void Remove(T entity)
    {
        dbSet.Remove(entity);
    }
    1 reference
    public void RemoveRange(IEnumerable<T> entity)
    {
        dbSet.RemoveRange(entity);
    }
}

```

Rysunek 4-12 Kod klasy `Repository` wraz z implementacją interfejsu `IRepository`. (źródło: opracowanie własne)

Klasa genetyczna „`Repository<T>`” implementuje ogólny interfejs „`IRepository`”, ogranicza typ „T” do klasy i umożliwia wykonanie podstawowych operacji CRUD (create, read, update, delete). Repozytorium będzie używane w celu dostępu do danych z poziomu aplikacji. Na samym początku tworzy się prywatne pole „`_db`” dające repozytorium dostęp do wcześniej utworzonego kontekstu bazy danych. Następne pole „`DbSet<T>`” reprezentuje zbiór encji w kontekście bazy danych. Kolejne elementy klasy wraz z ich charakterystyką wyglądają następująco:

- Konstruktor klasy „`Repository<T>`” przyjmuje obiekt „`ApplicationDbContext`” jako parametr i przypisuje go do pola `_db`. Następnie inicjalizuje pole `dbSet` jako zbiór danych typu T z bazy danych.
- Metoda „`Add`” dodaje nowy obiekt typu T do zbioru danych.

- Metoda „Get” przyjmuje wyrażenie lambda jako parametr i zwraca pierwszy obiekt typu T, który spełnia warunek wyrażenia.
- Metoda „GetAll” zwraca wszystkie obiekty typu T z zbioru danych jako kolekcję” I`Enumerable`<T>”.
- Metoda „Remove” usuwa podany obiekt typu T ze zbioru danych.
- Metoda „RemoveRange” usuwa podaną kolekcję obiektów typu T ze zbioru danych.

Klasa „Repository” jest kluczowa w budowanej warstwie dostępu do danych. Ona właśnie będzie dziedziczona przez kolejne interfejsy odpowiadające poszczególnym zbiorom danych. Klasa „BusStopRepository”, ukazana na rysunku „Implementacja repozytorium do obsługi danych dotyczących przystanków autobusowych” dziedziczy właśnie tę klasę i implementuje interfejs „IBusRepository”.

Zdefiniowany interfejs „IBusRepository” rozszerza się o interfejs „ IRepository” z określonym typem generycznym „BusStop”. Tak przygotowany interfejs można zaimplementować w konkretnej klasie, odpowiadającej za operacje na danym zbiorze danych, w tym przypadku na przystankach autobusowych.

```

2 references
public class BusStopRepository : Repository<BusStop>, IBusStopRepository
{
    private ApplicationDbContext _db;
    public BusStopRepository(ApplicationDbContext db) : base(db)
    {
        _db = db;
    }

    public void Update(BusStop obj)
    {
        _db.BusStops.Update(obj);
    }
}

3 references
public interface IBusStopRepository : IRepository<BusStop>
{
    void Update(BusStop obj);
}

```

Rysunek 4-13 Implementacja repozytorium do obsługi danych dotyczących przystanków autobusowych. (źródło: opracowanie własne)

Kolejna część kodu implementuje wzorzec repozytorium dla encji „BusStop”, co umożliwia wykonywanie operacji CRUD na tej właśnie encji, oddzielając logikę dostępu do danych od reszty aplikacji. Klasa „BusStopRepository” dziedziczy po klasie generycznej „Repository” z określonym typem „BusStop” i implementuje interfejs „IBusStopRepository”.

Prywatne pole „_db” typu „ApplicationDbContext” będzie używane do interakcji z bazą danych poprzez klasę kontekstu zarządzającą wcześniej określoną encją podczas wykonywania operacji na danych.

W następnie zdefiniowanym konstruktorze, który przyjmuje jako argument kontekst bazy danych, wywoływany jest konstruktor bazowy kontekstu bazy danych. W ciele

konstruktora „_db = db;” przypisuje przekazany obiekt kontekstu bazy danych do prywatnego pola „_db” umożliwiając w ten sposób dostęp do bazy danych w metodach tej klasy.

Metoda „Update” będąca elementem zaimplementowanego wzorca repozytorium przyjmuje obiekt typu „BusStop” jako argument i odpowiada za aktualizację danych dotyczących przystanków autobusowych. W ciele metody, wywoływana jest metoda „Update” na zbiorze „BusStops” w kontekście bazy danych „_db”, przekazywany jest do niej obiekt „obj” reprezentujący przystanek autobusowy. Działanie to spowoduje zaktualizowanie odpowiednich danych przystanku autobusowego w bazie danych.

Po przygotowaniu repozytorium dla danych „BusStop” należy je dodać do struktury jednostki pracy. Wzorzec „jednostka pracy” jest często stosowany w aplikacjach korzystających z dostępu do bazy danych, aby utrzymać spójność transakcji i zarządzać cyklem życia różnych operacji na danych w ramach jednej "jednostki pracy".

```
2 references
public class UnitOfWork : IUnitOfWork
{
    private ApplicationDbContext _db;
    9 references
    public IPassengerRepository Passenger { get; private set; }
    10 references
    public ITicketRepository Ticket { get; private set; }
    4 references
    public IDiscountRepository Discount { get; private set; }
    4 references
    public IBusStopRepository BusStop { get; private set; }

    0 references
    public UnitOfWork(ApplicationDbContext db)
    {
        _db = db;
        Passenger = new PassengerRepository(_db);
        Ticket = new TicketRepository(_db);
        Discount = new DiscountRepository(_db);
        BusStop = new BusStopRepository(_db);
    }

    8 references
    public void Save()
    {
        _db.SaveChanges();
    }
}

8 references
public interface IUnitOfWork
{
    9 references
    IPassengerRepository Passenger { get; }
    10 references
    ITicketRepository Ticket { get; }
    4 references
    IDiscountRepository Discount { get; }
    4 references
    IBusStopRepository BusStop { get; }

    8 references
    void Save();
}
```

Rysunek 4-14 Implementacja wzorca "jednostka pracy". (źródło: opracowanie własne)

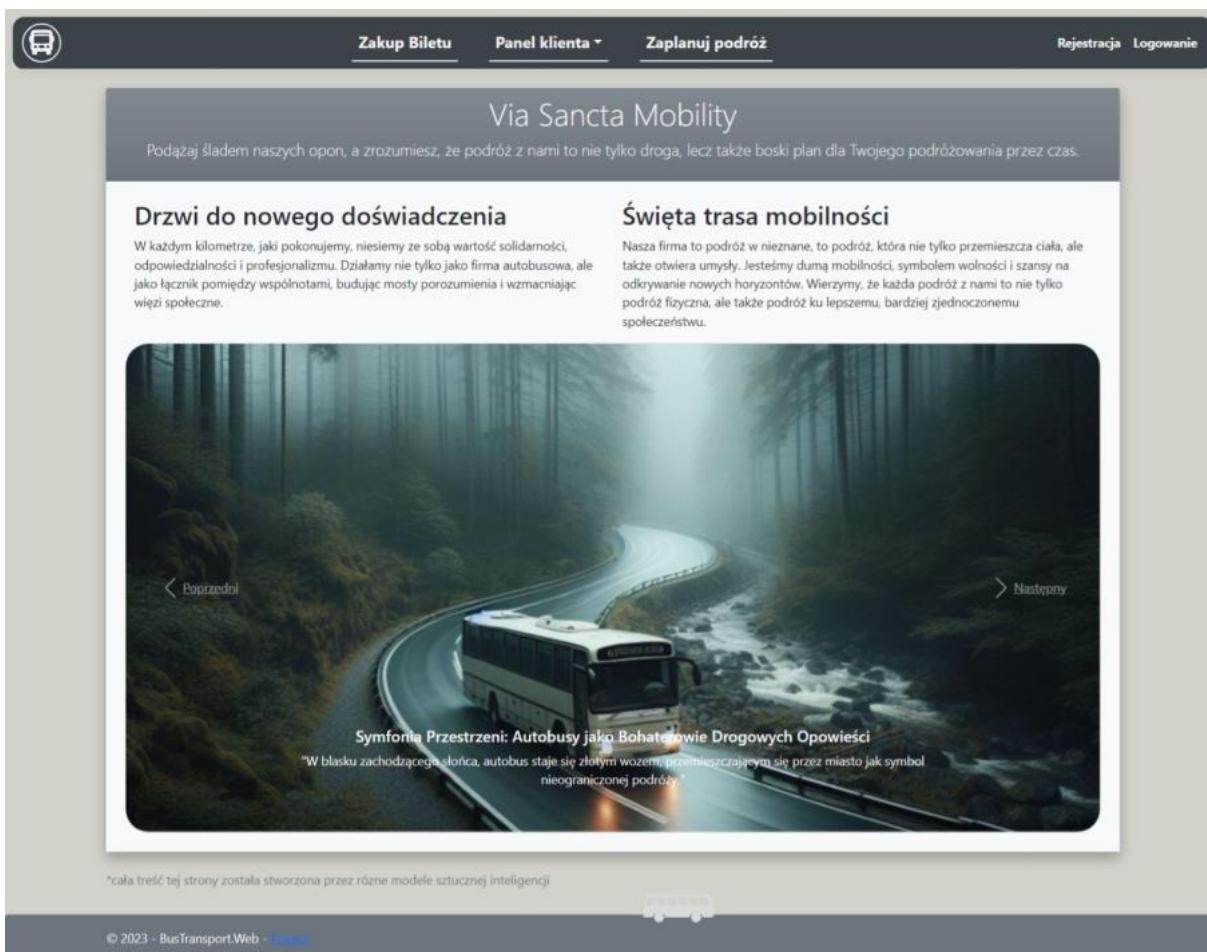
Interfejs „IUnitOfWork” określa strukturę z repozytoriami, na których ma pracować budowana aplikacja. Zdefiniowane są cztery różne repozytoria. Każda właściwość zwraca interfejs repozytorium, co pozwala na abstrakcję od konkretnych implementacji repozytoriów. Wszystkie te właściwości są tylko do odczytu, co oznacza, że można je tylko pobierać (nie ma możliwości ustawienia ich bezpośrednio z zewnątrz).

Metoda „Save” służy jako globalny sposób na zatwierdzanie wszystkich zmian dokonanych w ramach jednostki pracy. Zapisuje wszystkie zmiany w spójny i kontrolowany sposób, dzięki czemu można uniknąć częściowych aktualizacji danych, gdy operacje związane

z różnymi repozytoriami są zależne od siebie. Takie podejście pozwala na zarządzanie transakcjami i zapewnia spójność danych poprzez grupowanie operacji na danych w jednej jednostce pracy, co jest szczególnie przydatne w złożonych operacjach wymagających wielu kroków lub modyfikacji w różnych częściach modelu danych.

4.2.2. Strona główna serwisu

Strona główna stanowi domyślny, powitalny widok, który zobaczy osoba wchodząca na serwis internetowy. Jest on również wywoływany poprzez naciśnięcie loga w po lewej stronie paska nawigacyjnego. W tym przypadku jest to ikona autobusu wpisana w okrąg. Strona ta będzie przygotowana do wyświetlania ogólnych informacji o przewoźniku.



Rysunek 4-15 Widok strony głównej serwisu internetowego. (źródło: opracowanie własne)

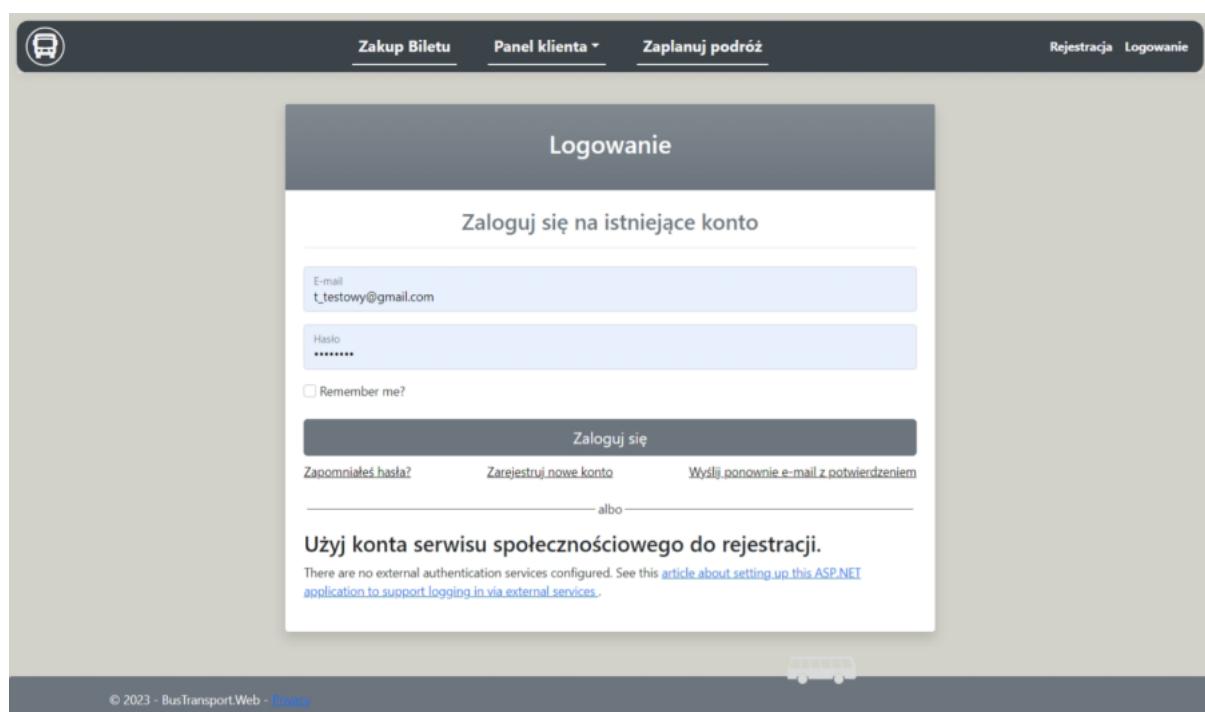
Niezależne od tego, czy użytkownik się zalogował, uzyska on dostęp do podstawowych funkcjonalności serwisu. Wtedy do dyspozycji zostaną oddane dwie, wyłączając logowanie i rejestrację, funkcjonalności: możliwość przeglądania informacji na temat przewoźnika na stronie głównej, oraz dostęp do zakładki „zaplanuj podróż”. Pozostałe dwa elementy paska

nawigacyjnego, „zakup biletu”, oraz panel klienta” będą niedostępne. Każdorazowo, kiedy użytkownik niezalogowany będzie próbował wejść z nimi w interakcję zostanie przeniesiony do strony logowania.

Przykładowa strona ukazana na rysunku „Widok strony głównej serwisu internetowego” została wypełniona treścią wygenerowaną modelem generującym tekst oraz grafikę na podstawie ogólnego opisu (tzw. „prompt”), wykorzystane narzędzie to: „Czat Bing”, „Bing Generator Obrazów”, „Dall-E”, „Midjourney”.

4.2.3. Logowanie i rejestracja

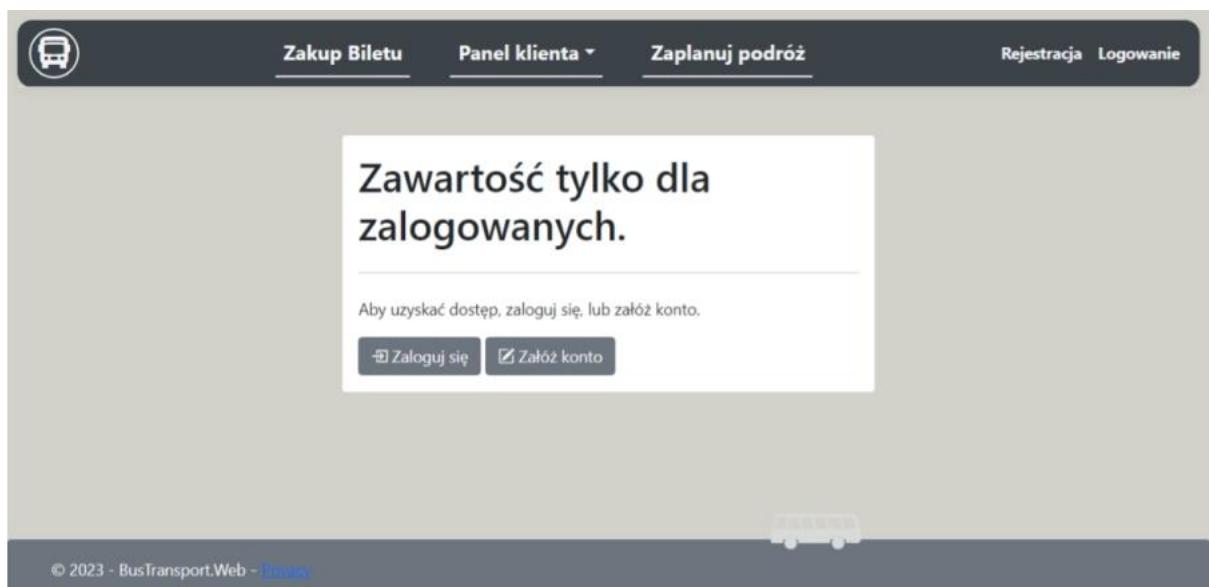
Funkcjonalności odpowiadające za logowanie, rejestracje i uwierzytelnianie użytkowników serwisu odpowiada przygotowany na podstawie dostarczonego przez Microsoft wraz ze środowiskiem .NET pakietu „.NET Identity”, moduł. Gotowe do konfiguracji w razie potrzeb rozwiązań pozwala programistom na zarządzanie tworzeniem użytkowników, uwierzytelnianiem, autoryzacją w bezpieczny i modułowy sposób. Wraz z pakietem uzyskiwany jest dostęp do wszystkich niezbędnych widoków i całej logiki związanej z nimi. Po ich konfiguracji budowane rozwiązania otrzymuje dostęp do funkcji, na przykład, rejestrowania się poprzez portal społecznościowy, wysłanie e-maila z weryfikacją, czy przypomnienie hasła.



Rysunek 4-16 Widok strony „logowanie” serwisu internetowego. (źródło: opracowanie własne)

Użytkownik, mając wcześniej założone konto, loguje się na nie wypełniając dwa pola: e-mail oraz hasło. Po udanej weryfikacji użytkownik uzyska dostęp do całej funkcjonalności serwisu internetowego. Jego e-mail jest widoczny w jasnoszarym polu po prawej stronie belki nawigacyjnej.

Użytkownik, w przypadku, gdy nie zalogował się do serwiska, a próbuje uzyskać dostęp do którejś z opcji dostępnej tylko dla zalogowanych, zostanie poinformowany o braku dostępu do niej oknem z odpowiednim komunikatem, co przedstawia poniższy rysunek „Widok strony z informacją o dostępności do jej funkcji”.



Rysunek 4-17 Widok strony z informacją o dostępności do jej funkcji. (źródło: opracowanie własne)

Formularz rejestracji posiada sześć wymaganych pól do wypełnienia. Każde z nich jest weryfikowane w trakcie wypełniania. E-mail musi zawierać znak „@” i domenę po nim, hasło należy zbudować z co najmniej sześciu znaków, jednego znaku specjalnego, wielkiej litery i cyfry. Świeżo zarejestrowany użytkownik jest automatycznie logowany na swoje nowo założone konto.

Rysunek 4-18 Widok strony "rejestracja" serwisu internetowego. (źródło: opracowanie własne)

System stwierdzając niespójność, czy błąd we wprowadzonych danych w którymś polu, powiadomi użytkownika o tym odpowiednim komunikatem pod tym polem. Przykład pokazuje rysunek „widok strony "rejestracja" serwisu internetowego”, gdzie stwierdzono dwa błędy we wpisanym haśle. Jest ono za krótkie i nie pasują do siebie.

4.2.4. Zakup biletu

Zalogowany pasażer ma dostęp do opcji zakupu biletu. Na samym początku, z listy przystanków pobieranej z bazy danych wybiera interesujący go przystanek początkowy, następnie przystanek końcowy.

Do komunikacji i wymiany danych został użyty wcześniej opisywany mechanizm oparty na dwóch wzorcach projektowych: repozytorium wraz z jednostką pracy. Rysunek „Kod kontrolera dla widoku "Zakup biletu"" pokazuje część kodu kontrolera odpowiadającego za logikę widoku „zakup biletu”.

Klasa „TicketPurchaseController” odpowiada za zachowanie widoku o nazwie „TicketPurchase”. Oba komponenty są współodpowiedzialne za wyświetlanie danych i operacji na nich. Kontroler korzysta z odpowiednich metod i właściwości klasy bazowej ułatwiających obsługę żądań HTTP i renderowanie widoków.

```

[Area(areaName: "Admin")]
1 reference
public class TicketPurchaseController : Controller
{
    private readonly IUnitOfWork _unitOfWork;
    0 references
    public TicketPurchaseController(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }

    0 references
    public IActionResult TicketPurchase()
    {
        TicketVM ticketVM = new()
        {
            DiscountList = _unitOfWork.Discount.GetAll().Select(u => new SelectListItem
            {
                Text = u.DiscountName,
                Value = u.Discount_Id.ToString()
            }),
            BusStopList = _unitOfWork.BusStop.GetAll().Select(z => new SelectListItem
            {
                Text = z.StopName,
                Value = z.BusStop_Id.ToString()
            }),
            Ticket = new Ticket()
        };
        return View(ticketVM);
    }
}

```

Rysunek 4-19 Część kodu kontrolera dla widoku "Zakup biletu". (źródło: opracowanie własne)

Konstruktor klasy „public TicketPurchaseController (IUnitOfWork unitOfWork)”, który przyjmuje instancję IUnitOfWork jako argument. Jest to przykład wstrzykiwania zależności, które pozwala na oddzielenie logiki biznesowej od konkretnych implementacji repozytoriów, ułatwiając testowanie i utrzymanie kodu.

Metoda „TicketPurchase()”, akcja kontrolera obsługująca wyświetlanie formularza zakupu biletów. Zwraca typ „IActionResult”, reprezentujący różne typy wyników akcji, takie jak wyświetlanie widoku, przekierowanie, zwrócenie pliku itp. W ciele metody tworzony jest obiekt „TicketVM” (ViewModel), który służy do przekazania danych między kontrolerem a widokiem. „TicketVM” zawiera dwie listy rozwijane (DiscountList, BusStopList) wypełnione danymi z repozytorium, pobranych przy użyciu wcześniej zdefiniowanego „_unitOfWork”. Następnie te dane zostają przekonwertowane na obiekty typu „SelectListItem”, które są używane w widokach do renderowania elementów typu <select>, które zostaną wyświetlone na liście, co pokazuje rysunek „Kod odpowiadający za elementy listy przystanków wraz z tą listą”. Na koniec metoda zwraca widok wraz z modelem „ticketVM” („return View(ticketVM)”), co umożliwia renderowanie formularza zakupu biletów z odpowiednimi danymi w widoku.

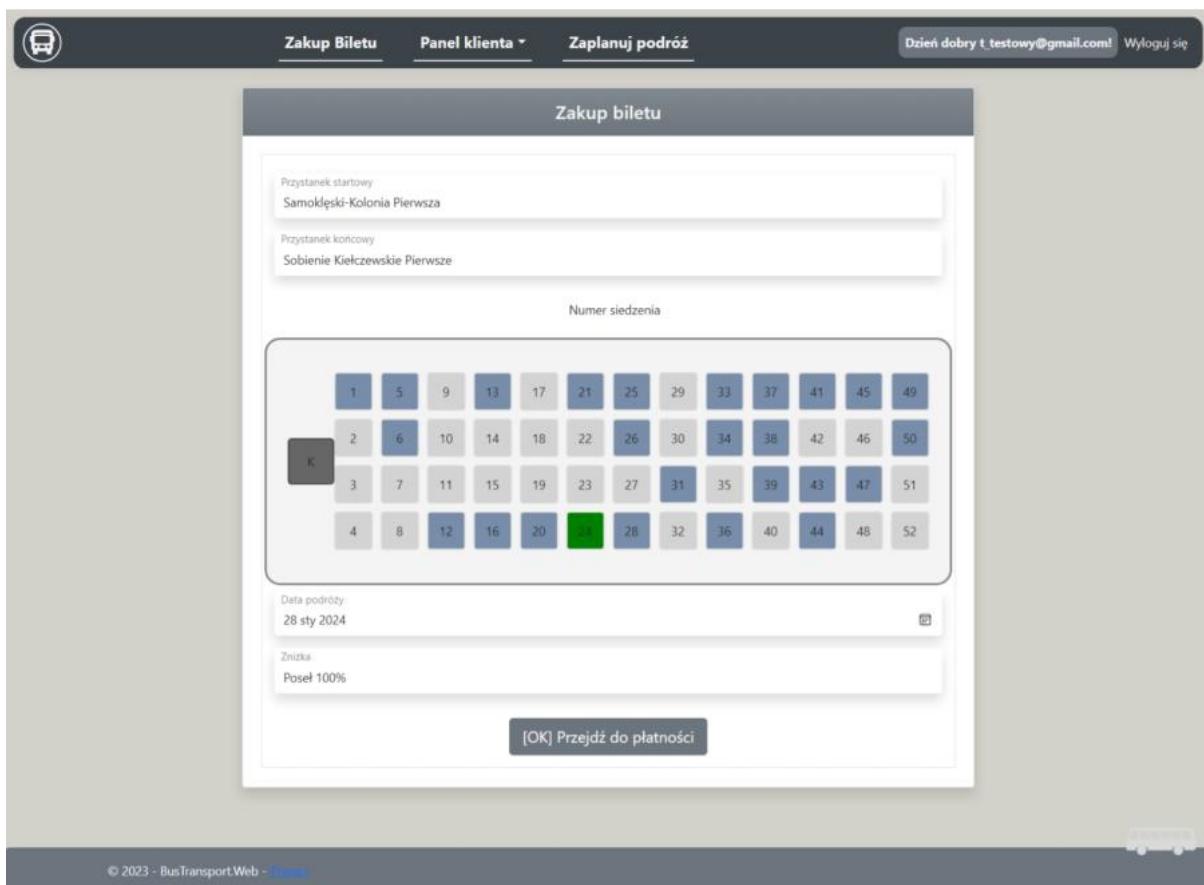
Prawa część poniższego rysunku pokazuje fragment kodu interfejsu użytkownika, po skonfigurowaniu kontrolera. Utworzone zostały dwie listy: dla przystanku startowego i końcowego wyświetlane po kliknięciu w pole wyboru.

The screenshot shows a user interface for buying a bus ticket. At the top, there are three tabs: "Zakup Biletu", "Panel klienta ▾", and "Zaplanuj podróż". Below this is a main title "Zakup biletu". A dropdown menu is open under the heading "Przystanek startowy". It contains a single item: "Warszawa". Below this, another dropdown menu is open under "Przystanek końcowy". It contains the message "-- Wybierz przystanek końcowy --" followed by a list of options: "Owsianka", "Rosochatka", "Samokleski-Kolonia Pierwsza", "Rozpuscie Pierwsze", "Niedźwida Duża", "Przeginia Duchowna", "Swędzieniewice", "Jeże", "Pędnikiaty", "Stare Pieścrogi", "Grabcze-Towarzystwo", "Przedmieście Szczebreszyńskie", "Sobienie Kielczewskie Pierwsze", "Stare Leśne Bohatery", "Warszawa", and "Kętna".

```
--> 14 <div class="form-floating py-2 col-12">
15   <select asp-for="@Model.BusStop.BusStop_Id" asp-items="@Model.BusStopList"
16     class="form-control border-0 shadow">
17     <option selected default>-- Wybierz przystanek startowy --</option>
18   </select>
19   <label class="ms-2">Przystanek startowy</label>
20   <span asp-validation-for="BusStop.BusStop_Id" class="text-danger"></span>
21 </div>
22 <div class="form-floating py-2 col-12">
23   <select asp-for="@Model.BusStop.BusStop_Id" asp-items="@Model.BusStopList"
24     class="form-control border-0 shadow">
25     <option selected default>-- Wybierz przystanek końcowy --</option>
26   </select>
27   <label class="ms-2">Przystanek końcowy</label>
28   <span asp-validation-for="BusStop.BusStop_Id" class="text-danger"></span>
29 </div>
```

Rysunek 4-20 Kod odpowiadający za elementy listy przystanków wraz z tą listą. (źródło: opracowanie własne)

Tag języka HTML „`<select>`” stanowi element wyboru z listy. Wartość tego pola powiązano z właściwością „`BusStop_Id`” obiektu „`BusStop`” w modelu przekazanym do widoku. Elementy listy rozwijanej będą pobierane z kolekcji `BusStopList` przygotowanej w modelu w kontrolerze (`asp-items="@Model.BusStopList"`). W obu listach ustawiono domyślną wartość, odpowiednio na: „wybierz przystanek startowy” i „wybierz przystanek końcowy”. Zasada działania obu list jest taka sama. Użytkownik wybiera przystanek początkowy i końcowy z dwóch takich samych list.



Rysunek 4-21 Widok poprawnie wypełnionego formularza kupna biletu. (źródło: opracowanie własne)

Poprawne sprecyzowanie przystanków umożliwi użytkownikowi wybór miejsca z interaktywnego schematu. Środkową część formularza zakupu biletu stanowi komponent z rozmieszczeniem siedzeń w autobusie. Przedstawia go rysunek „Widok poprawnie wypełnionego formularza kupna biletu”. Każde z nich ma przypisany numer. Na przedzie autobusu, po lewej stronie schematu znajduje się, oznaczone kolorem ciemnoszarym, siedzenie dla kierowcy. Zajęte miejsca system oznacza kolorem niebieskim, to wybranie przez pasażera podświetlono, poprzez kliknięcie myszką na nim, kolorem zielonym. Wybór miejsca siedzącego stanie się możliwy dopiero wtedy, gdy użytkownik wybierze przystanek początkowy i końcowy z rozwijanej listy. Do tego czasu wybór siedzenia będzie niemożliwy. Schemat, do czasu poprawnego wyboru przystanku, będzie nieaktywny, użytkownik nie będzie mógł wejść z nim w interakcję.

W polu pod schematem rozmieszczenie siedzeń wybierana jest data podnóża, pod nim, użytkownik wybiera jedną z obowiązujących zniżek na przejazd. Każda z nich jest, na podobnej zasadzie jak wcześniej opisywana lista przystanków, jest pobierana z odpowiedniej tabeli z bazy danych.

4.2.5. Panel klienta

Stanowi on środkową część górnej belki nawigacyjnej. Lista rozwijana, ukazująca się po kliknięciu w tę opcję, ukaże 3 kolejne funkcje: przeglądanie historii odbytych kursów zwrot biletu, twoje dane.

Pierwszy element menu, „historia kursów”, wyświetla zalogowanemu użytkownikowi tabele z historią odbytych przejazdów. Pasażer znajdzie tam informacje o wszystkich swoich wcześniejszych przejazdach. Tabela ukazuje datę podróży, miejscowości startową i końcową, imię i nazwisko pasażera oraz kwotę zapłaconą za bilet. Wykaz wszystkich przejazdów ukazuje rysunek „Widok wszystkich kursów zalogowanego pasażera”. W przypadku, gdy użytkownik nie ma żadnego przejazdu na swoim koncie, wyświetli się jedynie tabela z nagłówkiem, bez zawartości.

#	Data	Miejsce Początkowe	Miejsce Celowe	Imię i Nazwisko Pasażera	Kwota Biletu
1	2023-01-15	Warszawa	Kraków	T. Testowy	50.00 PLN
2	2023-02-20	Poznań	Wrocław	T. Testowy	40.00 PLN

Rysunek 4-22 Widok wszystkich kursów zalogowanego pasażera (źródło: opracowanie własne)

Głównym elementem menu „zwrot biletu” jest, podobna do poprzedniej, tabela z wykazem przejazdów. Zwrot biletu będzie możliwy na określony czas przed planowaną godziną odjazdu autobusu. Po jego odjeździe zwrot będzie niemożliwy, przycisk „zwrot biletu” będzie w tym przypadku nieaktywny, co ukazuje rysunek „Widok listy biletów zalogowanego użytkownika”. W przypadku istnienia możliwości zwrotu, pasażer zostanie, po naciśnięciu przycisku zwrotu, zostanie przekierowany na zewnętrzną stronę odpowiadającą za dostarczanie i obsługę metod płatności.

#	Data wyjazdu	Miejsce Początkowe	Miejsce Docelowe	Imię i Nazwisko Pasażera	Kwota Biletu	Akcja
1	2023-01-15	Warszawa	Kraków	T. Testowy	50.00 PLN	Zwrot Biletu
2	2023-02-20	Poznań	Wrocław	T. Testowy	40.00 PLN	Zwrot Biletu

Rysunek 4-23 Widok listy biletów zalogowanego użytkownika. (źródło: opracowanie własne)

Pozycja z rozwijanego menu „twoje dane” odpowiada za zarządzanie kontem. Istnieje możliwość zmiany hasła, ukazana na rysunku „Wybrana opcja zmiany hasła w panelu zarządzanie kontem pasażera”, zmiana e-maila, czy całkowite skasowanie konta. W tej sekcji aplikacji przeglądarkowej, dane podane przy rejestracji mogą zostać zmienione, w ostatniej zakładce w menu po lewej stronie „Dane osobiste” istnieje możliwość całkowitego usunięcia konta użytkownika.

Zakup Biletu
Panel klienta ▾
Zaplanuj podróż

Dzień dobry t_testowy@gmail.com!
Wyloguj się

Zarządzaj swoim kontem

Zmień ustawienia swojego konta

[Profil](#)
[E-mail](#)
[Hasło](#)
[Weryfikacja dwuetapowa](#)
[Dane osobiste](#)

Zmień hasło

Aktualne hasło

Nowe hasło

Powtórz nowe hasło

Nowe hasła do siebie nie pasują

Zaktualizuj hasło

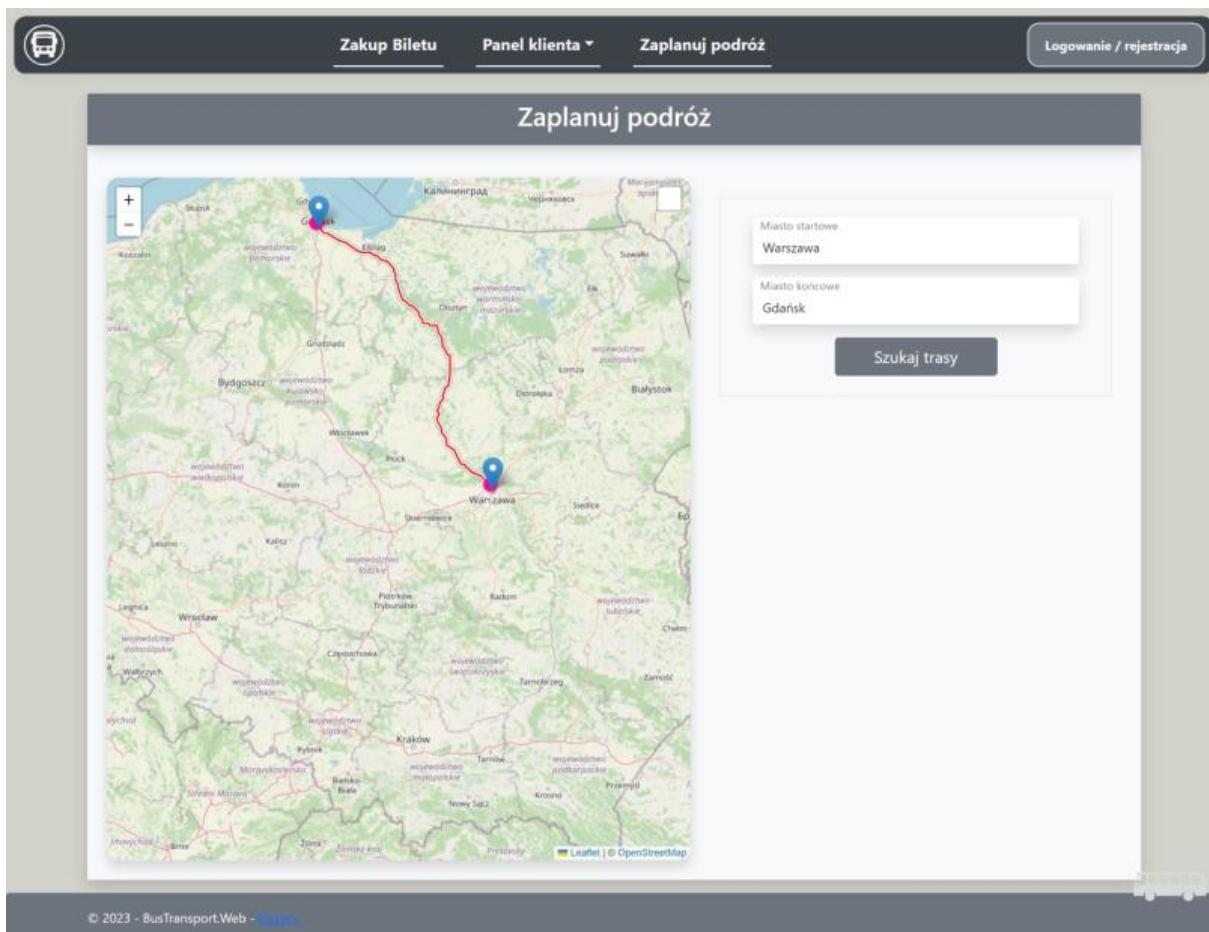
© 2023 - BusTransportWeb - [Privacy](#)

Rysunek 4-24 Wybrana opcja zmiany hasła w panelu zarządzanie kontem pasażera. (źródło: opracowanie własne)

Rysunek „Wybrana opcja zmiany hasła w panelu zarządzanie kontem pasażera” przedstawia menu odpowiadające za zmianę hasła wraz z powiadomieniem o jego nieprawidłowości. Wykorzystanie i skonfigurowane rozwiązanie .NET Identity dostarcza również możliwość połączenia konta serwisu internetowego z kontem założonym na serwisach społecznościowych.

4.2.6. Zaplanuj podróż

Funkcjonalność dostępna dla użytkownika zalogowanego i niezalogowanego. Na podstawie wartości wpisanej w pola „przystanek początkowy” i „przystanek końcowy” system wyświetla trasę przejazdu, analizując wcześniej, czy taka trasa jest dostępna w ofercie przewoźnika.



Rysunek 4-25 Widok strony "zaplanuj podróż" serwisu internetowego. (źródło: opracowanie własne)

Zaznaczana jest ona na mapie czerwoną linią, wraz ze wcześniej określonymi punktami początkowym i końcowym podróży. Te są oznaczone niebieskimi pineskami. W przypadku braku występowania trasy, którą zainteresował się pasażer system wyświetla odpowiedni komunikat.

Mapę stanowi OpenStreetMap, darmowe, otwartoźródłowe, tworzone przez społeczność rozwiązywanie dostarczające mapę świata wraz z punktami użyteczności, POI – points of interests.. Za zachowanie mapy, jej działanie i interakcję odpowiada biblioteka języka JavaScript Leaflet – narzędzie umożliwiające integrowanie interaktywnych map na stronach internetowych. [16] [17] [18]

4.3. Implementacja aplikacji mobilnej

Druga część systemu wspomagającego pracę krajowego transportu autobusowego to aplikacja mobilna dla kierowcy. Korzysta z tej samej, wspólnej dla całego systemu bazy danych. Została stworzona do użytkowania w systemie Android, we framework'u MAUI będącym częścią środowiska .NET.

Struktura domyślna plików aplikacji budowanej w MAUI jest bardzo podobna do ASP.NET. W pliku MauiProgram.cs znajduje się większość konfiguracji aplikacji. Rejestrowane są w nim, poprzez wstrzykiwanie zależności, kolejne usługi mające ubogacać budowaną aplikację.

4.3.1. Wykorzystane wzorce i podejścia do stworzenia aplikacji mobilnej

Aplikacja mobilna dla kierowcy powstała w oparciu o wzorzec MVVM, model–view–viewmodel oddzielający interfejs graficzny od warstwy odpowiedzialnej za logikę wymiany danych. Ogólne jego założenia są podobne do wykorzystanego wcześniej MVC. Modelem jest najczęściej wcześniej przygotowana encja. Widok komunikuje się z „ViewModelem”, który pośredniczy między widokiem a modelem. ViewModel Przechowuje potrzebne dane, związane z interfejsem użytkownika, do wyświetlenia interfejsu, pobrane z bazy danych lub innych źródeł. Ponadto obsługuje też interakcje użytkownika jak kliknięcie przycisków, wprowadzanie tekstu, po czym uruchamiane zostają odpowiednie działania lub odpowiednio aktualizowane są dane.



```
5 <Shell
6   x:Class="BusTransport.Mob.AppShell"
7   xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
8   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
9   xmlns:local="clr-namespace:BusTransport.Mob"
10  xmlns:pages="clr-namespace:BusTransport.Mob.Views"
11  Title="BusTransport.Mob" BackgroundColor="LightGrey" DisabledColor="#BDBDBD"
12  ForegroundColor="#000000" Shell.FlyoutBehavior="Disabled" Shell.NavBarIsVisible="False"
13  TabBarBackgroundColor="#FFFFFF" TabBarForegroundColor="#0078D7" TabBarTitleColor="#0078D7"
14  TabBarUnselectedColor="#656565" TitleColor="Black" UnselectedColor="#656565">
15
16  <TabBar>
17    <Tab Title="Trasa" Icon="map.png">
18      <ShellContent ContentTemplate="{DataTemplate pages:RoutePage}" />
19    </Tab>
20    <Tab Title="Mapa autobusu" Icon="seat.png">
21      <ShellContent ContentTemplate="{DataTemplate pages:BusMapPage}" />
22    </Tab>
23    <Tab Title="Lista przystanków" Icon="stop.png">
24      <ShellContent ContentTemplate="{DataTemplate pages:StopListPage}" />
25    </Tab>
26    <Tab Title="Kasowanie biletu" Icon="scan.png">
27      <ShellContent ContentTemplate="{DataTemplate pages:TicketValidationPage}" />
28    </Tab>
29  </TabBar>
30</Shell>
```

Rysunek 4-26 Plik AppShell.xaml stanowiący dolną belkę nawigacyjną w aplikacji (źródło: opracowanie własne)

Plik AppShell.xaml określa podstawową strukturę interfejsu użytkowania. Pełni rolę kontenera dla aplikacji upraszczając jej tworzenie poprzez zarządzanie nawigacją poprzez odpowiednie komponenty. W budowanej aplikacji mobilnej dla kierowcy AppShell odpowiada za dolną belkę nawigacyjną. Stanowią ją cztery zakładki: „Trasa”, „Mapa autobusu”, „Lista przystanków”, „Kasowanie biletu”.

Kod z konfiguracją paska nawigacyjnego z zakładkami pokazuje rysunek „Plik AppShell.xaml stanowiący dolną belkę nawigacyjną w aplikacji”. W języku XAML każdy z tagów ma swoje właściwości. Odpowiadają one za zachowanie komponentu, jego wygląd, czy pozwalają się połączyć z logiką pisana w języku c#. Są wyróżnione kolorem pomarańczowym, a ich wartości ujęte w cudzysłowie. Sam początek pliku tag „Shell” konfiguruje ogólny wygląd paska nawigacyjnego, takie jak tło całego paska, poszczególnych przycisków, a właściwość Shell.NavBarIsVisible="False" ukrywa górny pasek nawigacyjny systemu Android.

W następnej sekcji kodu, „<TabBar>” definiuje elementy typu „<Tab>” reprezentujące zakładki w aplikacji. Te elementy będą zawsze widoczne w trakcie przełączanie się między nimi. Każdy element „<ShellContent>” wewnątrz „<Tab>” poprzez właściwość „ContentTemplate="{DataTemplate pages:RoutePage}"" odnosi się do konkretnej, wcześniej przygotowanej strony i odpowiada za jej wyświetlenie. „Pages” jest przestrzenią nazw dodaną w pierwszej sekcji kodu i stanowi ona odwołanie do katalogu „Views”, w którym znajdują się wszystkie widoki wykorzystywane w aplikacji. Każda zakładka ma swoją nazwę (właściwość „Title”), oraz ikonę wyświetlana nad nazwą (właściwość „Icon”) zaczerpniętą z jednego z domyślnie tworzonych przez środowisko katalogu „Images” w folderze „Resources”.

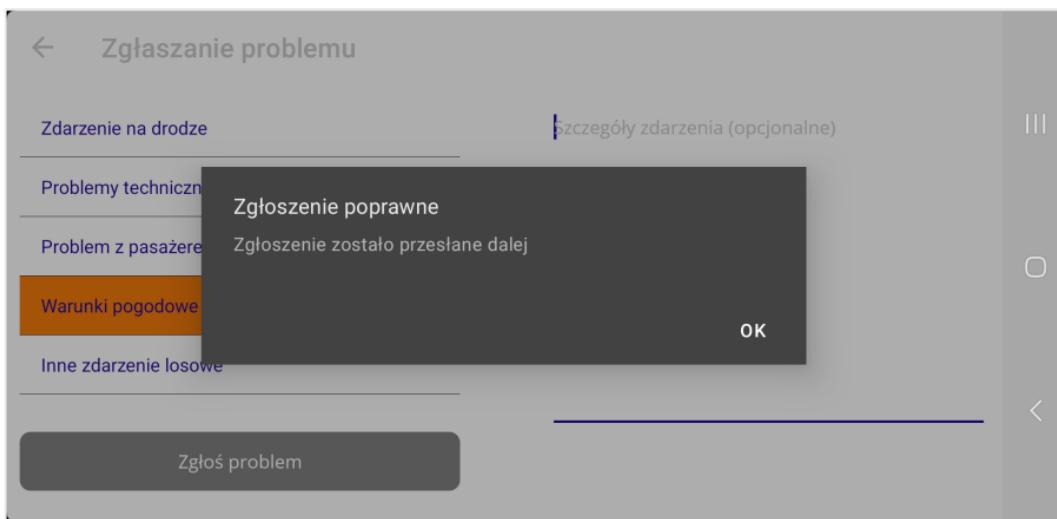
4.3.2. Logowanie i zgłaszanie problemu

Okno logowania jest pierwszym, ukazującym się po uruchomieniu aplikacji. Kierowca wprowadza tutaj poznane wcześniej login wraz z hasłem, założone przez przełożonego w przedsiębiorstwie autobusowym. Nie będąc zalogowanym ma możliwość zgłoszenia problemu naciskając przycisk „zgłoś problem”. Pomyślne zalogowanie przekieruje użytkownika do głównego widoku aplikacji.



Rysunek 4-27 Widok logowania w aplikacji dla kierowcy (źródło: opracowanie własne)

Formularz zgłoszenia jest wywoływany poprzez dotknięcie przycisku „zgłoś problem”. Wyświetlane jest wtedy nowe okno, pokazane na rysunku. Po lewej jego stronie zgłaszający ma dostęp do przewijanej listy z ogólnymi kategoriami zgłoszeń. Po prawej pole tekstowe do opisania szczegółów zastanego zdarzenia.



Rysunek 4-28 Formularz zgłaszczenia problemu w aplikacji dla kierowcy (źródło: opracowanie własne)

Omawiane okno prezentuje rysunek „Formularz zgłaszczenia problemu w aplikacji dla kierowcy”. Pomyślnie wysłany formularz zgłoszenia zakomunikuje odpowiednie wyskakujące okienko. Zostaje on wtedy przekazany osobie mającej dostęp do wszystkich zgłoszeń i mogącej je dalej przekazywać, na przykład służbom porządkowym, czy na ich podstawie robić raporty i statystyki.

4.3.3. Widok aktualnej trasy

Główny, domyślny widok po zalogowaniu się kierowcy do aplikacji. Ukazuje mapę z linią ciągłą wytyczającą realną trasę, jaką będzie podążał pojazd. Punktami na mapie oznaczona przystanki. Widok można w każdej chwili skalować, by przystosować go do aktualnych potrzeb.



Rysunek 4-29 Okno aktualnego przebiegu trasy w aplikacji dla kierowcy (źródło: opracowanie własne)

Zaczynając implementację obsługi map w aplikacji należy pobrać paczkę NuGet o nazwie „Mapsui.Maui”. Umożliwia dokonywanie operacji geometrycznych, nanoszenie dodatkowych powłok na mapę, czy korzystanie z różnych dostawców map, takich jak Google Maps lub OpenStreetMap. Tak jak w serwisie internetowym dla pasażera, w tym przypadku mapę dostarcza otwartoźródłowe rozwiązań „OpenStreetMap”.

4.3.4. Mapa autobusu

Mapa autobusu w rzeczywisty sposób odzwierciedla zajętość miejsc przez pasażerów. Każdorazowo, gdy kierowca kasuje bilet jest „zajmowane” odpowiednie miejsca wcześniej wybrane przez pasażera, na schemacie miejsc, w trakcie kupowania biletu.



Rysunek 4-30 Zajętość miejsc w autobusie (źródło: opracowanie własne)

Kolor jasnoniebieski ukazuje miejsca już zajęte, kolor szary te, które są jeszcze dostępne. Zajęcie odbywa się poprzez zeskanowanie kodu QR na bilecie. Po „skasowaniu biletu” w aplikacji podświetla się miejsce, na które powinien się udać pasażer, po wyjściu pasażera na przystanku docelowym, miejsce zmienia kolor na szary, staje się znowu dostępne dla innego pasażera.

4.3.5. Lista przystanków

Lista przystanków odzwierciedla trasę pokazaną w pierwszej zakładce „trasa”. Tutaj ukazana jest jako lista przystanków ułożonych w kolejności odpowiadającej ułożeniu ich na trasie. Na podstawie wcześniej przydzielonego trasy do kursu i kursu dla kierowcy rozkład będzie ściągany z bazy danych po zalogowaniu się kierowcy do aplikacji.

Przystanek:	Pruszków	Przyjazd: 08:35	Odjazd: 08:40
Przystanek:	Żyrardów	Przyjazd: 09:10	Odjazd: 09:15
Przystanek:	Skierniewice	Przyjazd: 09:45	Odjazd: 09:50
Przystanek:	Łowicz	Przyjazd: 10:20	Odjazd: 10:25
Przystanek:	Sieradz	Przyjazd: 10:55	Odjazd: 11:00
Przystanek:	Wieluń	Przyjazd: 11:30	Odjazd: 11:35

Below the table are four navigation icons: 'Trasa' (route), 'Mapa autobusu' (bus map), 'Lista przystanków' (list of stops), and 'Kasowanie biletu' (ticket cancellation).

Rysunek 4-31 Lista przystanków aktualnie odbywającego się kursu (źródło: opracowanie własne)

Tabela została podzielona na trzy kolumny: tę po prawej stronie okna aplikacji, pokazującą miejscowościę, bądź nazwę przystanku, środkową z godziną przyjazdu i lewą z godziną odjazdu. Poniższy obrazek przedstawia kod w języku XAML, który odpowiada za wizualną stronę tabeli z listą przystanków, oraz określa ścieżkę dostępu do danych we właściwości „ItemsSource”.



```

2  <ContentPage
3      x:Class="BusTransport.Mob.Views.StopListPage"
4      xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
5      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
6      xmlns:local="clr-namespace:BusTransport.Mob.ViewModels"
7      Title="Lista przystanków">
8          <ScrollView>
9              <StackLayout Padding="20">
10                 <CollectionView Margin="10" BackgroundColor="#F0F0F0" ItemsSource="{Binding BusStops}">
11                     <CollectionView.ItemTemplate>
12                         <DataTemplate>
13                             <Grid Padding="10" BackgroundColor="#FFFFFF" ColumnDefinitions="*,*,*" ColumnSpacing="15">
14                                 <RowDefinitions>
15                                     <RowDefinition Height="*"/>
16                                     <RowDefinition Height="2" />
17                                 </RowDefinitions>
18                                 <Label Grid.Row="0" Grid.Column="0" FontAttributes="Bold" FontSize="10" HorizontalTextAlignment="Start" Text="Przystanek:" TextColor="#333333" VerticalOptions="Center" />
19                                 <Label Grid.Row="0" Grid.Column="0" FontAttributes="Bold" FontSize="16" HorizontalTextAlignment="End" Text="{Binding StopName}" TextColor="#333333" VerticalOptions="Center" />
20                                 <BoxView Grid.Row="1" Grid.Column="0" BackgroundColor="#333333" HeightRequest="1" HorizontalOptions="Fill" />
21                                 <Label Grid.Row="0" Grid.Column="1" Margin="40,0,0,0" FontAttributes="Bold" FontSize="10" HorizontalTextAlignment="Start" Text="Przyjazd:" TextColor="#333333" VerticalOptions="Center" />
22                                 <Label Grid.Row="0" Grid.Column="1" FontSize="16" HorizontalOptions="Center" Text="{Binding Arrival, StringFormat'{0:hh\\::mm}'}" TextColor="#666666" VerticalOptions="Center" />
23                                 <BoxView Grid.Row="1" Grid.Column="1" BackgroundColor="#666666" HeightRequest="1" HorizontalOptions="Fill" />
24                                 <Label Grid.Row="0" Grid.Column="2" Margin="40,0,0,0" FontAttributes="Bold" FontSize="10" HorizontalTextAlignment="Start" Text="Odjazd:" TextColor="#333333" VerticalOptions="Center" />
25                                 <Label Grid.Row="0" Grid.Column="2" FontSize="16" HorizontalTextAlignment="Center" Text="{Binding Departure, StringFormat'{0:hh\\::mm}'}" TextColor="#666666" VerticalOptions="Center" />
26                                 <BoxView Grid.Row="1" Grid.Column="2" BackgroundColor="#666666" HeightRequest="1" HorizontalOptions="Fill" />
27                         </Grid>
28                     </DataTemplate>
29                 </CollectionView.ItemTemplate>
30             </CollectionView>
31         </StackLayout>
32     </ScrollView>
33 </ContentPage>

```

Rysunek 4-32 Kod w języku XAML odpowiadający za wizualną stronę listy przystanków

Tag otwierający „ContentPage” to element interfejsu wyświetlający treść we wcześniej ustalonym układzie jak stos czy siatka. Zawarty w nim, „ScrollView” umożliwia przewijanie treści w nim zawartej, wykraczającej poza obszar. „ScrollView” otacza element „StackLayout” pozwalający rozmieszczać elementy układając je jeden nad drugim lub obok siebie. Potem następuje przygotowanie komórki odpowiadającej za wyświetlanie odpowiednich danych. W „CollectionView” wyświetlana będzie kolekcja danych w uporządkowany i konfigurowalny sposób, zaś „CollectionView.ItemTemplate” definiuje układ dla każdego elementu zawartego w „CollectionView”. Pojedyncza komórka składać się będzie z siatki („Grid”) z dwoma wierszami i trzema kolumnami. Każda z nich ma określoną wysokość, lub szerokość, które można określić niezamienialną wartością wyrażoną w pikselach, lub „gwiazdką”. W „Grid.RowDefinitions” określana jest ilość wierszy siatki. W tym przypadku przewidziane będą dwie: pierwsza mająca wysokość jednej gwiazdki, druga dwóch. Oznacza to, że obszar całej siatki jest dzielony na 3 równe części, z czego jedna z nich przypada pierwszemu wierszowi, a kolejne dwie drugiemu. Używając gwiazdek przy określaniu szerokości i wysokości siatki, ale też i innych komponentów, w języku XAML mamy pewność, że przy zmieniającym się

rozmiarze okna, te komponenty zawsze się będą do niego dopasowywać. Wiersz listy przystanków składa się z kilku elementów: etykiet z nazwą przystanku i godzinami odjazdu i przyjazdu, elementami „BoxView” który te etykiety podkreśla czarnym kolorem. Poprzez połączenie ze źródłem danych właściwością „Text” dla każdej etykiety, mogą one wyświetlać dowolne zbiory danych odpowiadające typom i strukturom użytym podczas konfigurowania całej listy. [19]

4.3.6. Skanowanie biletów

Po wywołaniu funkcji „kasowanie biletu” system uruchomi aparat urządzenia. Ono zaś zgłosi gotowość do skanowania biletów poprzez komunikat „kasowanie biletów możliwe”. Użytkownik skieruje urządzenie na kod QR pokazany przez pasażera na jego bilecie. System wyświetli komunikat o pomyślnym skasowaniu biletu, wraz z podstawowymi jego dennymi, umożliwiającymi podstawą, zgodną z aktualnym prawem, identyfikację pasażera w autobusie przez kierowcę.

```
4  <ContentPage
5    x:Class="BusTransport.Mob.Views.TicketValidationPage"
6    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
7    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
8    xmlns:zxing="clr-namespace:ZXing.Net.Maui.Controls;assembly=ZXing.Net.MAUI.Controls"
9    Title="Kasowanie biletu"
10
11    <zxing:CameraBarcodeReaderView x:Name="barcodeReader" BarcodesDetected="barcodeReader_BarcodesDetected" />
12  </ContentPage>
```

Rysunek 4-33 Kod w języku XAML dodający komponent z oknem aparatu. (źródło: opracowanie własne)

Za skanowanie kodów QR odpowiada jedno z rozwiązań przygotowanych przez społeczność .NET MAUI – biblioteka Zebra Crossing (ZXing). Pakiet „ZXing.Net.Maui.Control” instaluje się w Menedżerze Pakietów NuGet. By móc z niej korzystać, na samym początku trzeba uzyskać zezwolenie urządzenia na wykorzystywanie aparatu. W widoku, który ma odpowiadać za użyteczność aparatu, należy dodać jedną linijkę z kodem, która zainicjuje komponent, określi jego nazwę, oraz utworzy metodę wywoływaną po wykryciu kodu QR. Przedstawia to linijka 11 kodu zawartego w obrazku „Kod w języku XAML dodający komponent z oknem aparatu”.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android">
3    <application android:allowBackup="true" android:icon="@mipmap/appicon" android:supportsRtl="true"></application>
4    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
5    <uses-permission android:name="android.permission.INTERNET" />
6    <uses-permission android:name="android.permission.CAMERA" />
7    <uses-sdk />
8  </manifest>
```

Rysunek 4-34 Zawartość pliku AndroidManifest.xml. (źródło: opracowanie własne)

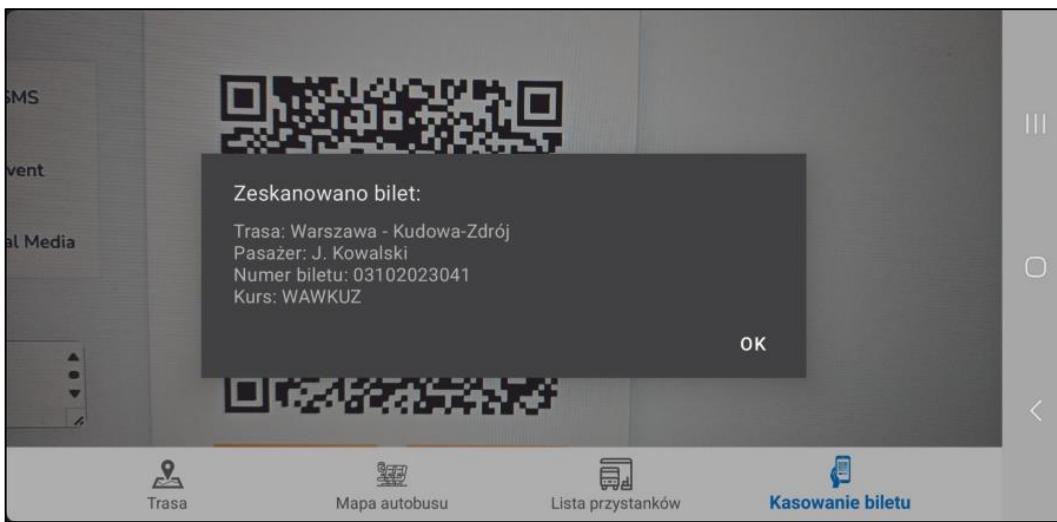
W katalogu Platforms/Android znajduje się plik konfiguracyjny AndroidManifest.xml, w nim właśnie nadajemy budowanej aplikacji odpowiednie uprawnienia, czy określamy docelową wersję systemu, na którą ma być dostępny aplikacja. By móc korzystać z aparatu urządzenia, należy do niego dodać linijkę zaznaczoną na niebiesko na obrazku „Zawartość pliku AndroidManifest.xml”.

```
6 references
3 public partial class TicketValidationPage : ContentPage
4 {
5     private bool isProcessingScan = false;
6     1 reference
7     public TicketValidationPage()
8     {
9         InitializeComponent();
10        barcodeReader.Options = new ZXing.Net.Maui.BarcodeReaderOptions
11        {
12            Formats = ZXing.Net.Maui.BarcodeFormat.QrCode,
13            AutoRotate = true,
14            Multiple = false
15        };
16        barcodeReader.CameraLocation = ZXing.Net.Maui.CameraLocation.Rear;
17    }
18
19     0 references
20     private void barcodeReader_BarcodesDetected(object sender, ZXing.Net.Maui.BarcodeDetectionEventArgs e)
21     {
22         if (isProcessingScan) return;
23
24         var first = e.Results?.FirstOrDefault();
25         if (first is null) return;
26
27         isProcessingScan = true;
28
29         Dispatcher.DispatchAsync(async () =>
30         {
31             await DisplayAlert(title: "Zeskanowano bilet:", first.Value, cancel: "OK");
32             isProcessingScan = false;
33         });
34     }
35 }
```

Rysunek 4-35 Kod konfigurujący aparat. (źródło: opracowanie własne)

Implementacja funkcjonalności skanera kodów QR odbywa się w konstruktorze „TicketValidationPage”, pokazuje to obrazek „Kod konfigurujący aparat”. Na samym początku, w linijce 5 inicjalizuje się zmienną typu bool odpowiadającą za śledzenie stanu procesu skanowania, czy ten jest w toku, czy nie. Następnie, w linijce 9, konfigurowane są podstawowe opcje działania aparatu: format kodu jaki ma być rozpoznawany, automatyczne obracanie okna z obrazem z kamery i możliwość rozpoznawania kilku kodów jednocześnie.

Kluczowym elementem w działaniu skanera kodów jest zdarzenie „barcodeReader_BarcodesDetected”. Na samym jego początku sprawdzany jest status działania skanera, jeżeli wcześniej rozpoczęty, a proces się nie zakończył zdarzenie kończy swoje działanie. W przeciwnym wypadku utworzona zostanie zmienna przechowująca treść kodu QR – „first”. Po wykryciu kodu „isProcessingScan” ustawiane jest na „true” aby wskazać, że rozpoczyna się przetwarzanie skanowania.



Rysunek 4-36 Okno aparatu odpowiedzialne za skanowanie kodu QR (źródło: opracowanie własne)

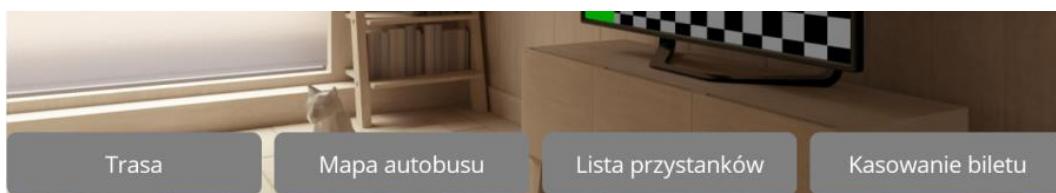
Następnie wywoływana jest operacja asynchroniczna wyświetlająca okno pokazujące treść o zeskanowanego kodu QR, w tym przypadku, przykładowego biletu, co widać na powyższym rysunku pokazującym szczegółową treść kodu QR będącego na potencjalnym bilecie. [19] [18] [14]

5. PODSUMOWANIE I WNIOSKI

Postawiony wcześniej cel pracy inżynierskiej: „opracowanie projektu i implementacja wybranych funkcjonalności systemu wspomagającego pracę krajowego transportu autobusowego” został zrealizowany. Powstało oprogramowanie składające się z dwóch komponentów: serwisu internetowego dla pasażera i aplikacji mobilnej dla kierowcy. Część przedstawionych wcześniej pomysłów nie została zaimplementowana w ogóle, na przykład: w rozdziale poświęcanym analizie przedstawiona jest rola w systemie „koordynator tras i przejazdów”. W toku projektu i późniejszej implementacji został on, jak i potencjalny moduł obsługi zgłaszanych zdarzeń pominięty, ze względu na skomplikowanie dużego podsystemu i czas, jaki trzeba by było na niego poświęcić.

Z rzeczy zaimplementowanych częściowo warto wspomnieć o aplikacji mobilnej i mapie z trasą. Pojawiło się dużo problemów ze spójnością danych przestrzennych. Domyślnie, poprzez Entity Framework i pakiet NetTopologySuite, dane są mapowane na jeden z typów obsługiwanych przez SQL Server – geography, bez problemu działa na mapie używanej w serwisie internetowym, w aplikacji mobilnej już nie. Framework .NET MAUI jest dosyć nowy, przez co wiele kwestii wymaga uzupełnienia i dopracowania przez twórcę. Udało się znaleźć kilka rozwiązań stworzonych przez społeczność, ale żadne nie pozwalało na to, by obie aplikacje mogły pracować na tych samych danych, z tego samego źródła.

Inną, nie do końca zaimplementowaną funkcjonalnością, również w przypadku aplikacji mobilnej, jest skanowanie kodów QR na biletce. Obecnie, po pierwszym wejściu w zakładkę „Kasowanie biletu” funkcja działa prawidłowo. Po zmianie na dowolną z trzech pozostałych zakładek i powrocie do skanowania ukaże się jedynie czarny ekran. Na samym początku powstała inna wersja dolnej belki nawigacyjnej. Zamiast, opisywanego w poprzednim rozdziale „AppShell” i wraz z nim „TabBar” wykorzystano przyciski uporządkowane w siatce i zakotwiczone na stałe u dołu ekranu. Tutaj z kolei aparat działał, nawet po zmianie okna na inny i powrocie do aparatu, ale skanowanie nie odbywało się prawidłowo, choć wykorzystywana była ta sama logika co wcześniej. Poprzednią wersję interfejsu pokazuje poniższy rysunek.



Rysunek 5-1 Pierwsza wersja belki nawigacyjnej w aplikacji mobilnej (źródło: opracowanie własne)

Perspektyw do rozwoju aplikacji w przyszłości jest kilka. Na pewno warto zaimplementować wcześniej wspomniany moduł do obsługi zdarzeń dla koordynatora. Należy też wyeliminować wspomniane błędy, w kliku miejscach poprawić interfejs użytkownika. W aplikacji przeglądarkowej, kontrolki górnej belki nawigacyjnej czasami delikatnie drżą przy najeżdżaniu kursorem na nie. Problem ten powinny rozwiązać poprawki w plikach stylów kaskadowych odpowiedzialnych za tę część strony. Dobrym wyjściem też byłoby dostosowanie systemu tak, by potencjalny użytkownik, przedsiębiorstwo zajmujące się transportem autobusowym, mogło je w pełni konfigurować: dodawać swoje grafiki na stronie, zarządzać flotą i ją kontrolować poprzez odpowiednie moduły itp.

Przedstawiony powyżej, w czterech rozdziałach, proces powstawania oprogramowania dla transportu autobusowego jest dobrym wyjściem do dalszego jego rozwoju. Powstały solidne ramy i perspektywy do budowy jednego systemu, który wspomoże, ostatnio odrobinę zapomniany, krajowy, międzymiastowy transport autobusowy.

Bibliografia

- [1] R. Khan, „Steppe 1.0, Going Nomad. The early Indo-Europeans' great leap forward,” 8 Maj 2021. [Online]. Available: <https://razib.substack.com/p/steppe-10-going-nomad>. [Data uzyskania dostępu: 1 Maj 2023].
- [2] LANtaNews, „The History of the Bus,” LANtaNews, 16 Maj 2012. [Online]. Available: <http://lantanews.blogspot.com/2012/05/history-of-bus.html>. [Data uzyskania dostępu: 1 Maj 2023].
- [3] J. Snowden, „What were horsecars?,” 9 Maj 2014. [Online]. Available: <https://www.historyanswers.co.uk/inventions/what-were-horsecars/>. [Data uzyskania dostępu: 1 Maj 2023].
- [4] SFMTA, „Cable Car History,” [Online]. Available: <https://www.sfmta.com/getting-around/muni/cable-cars/cable-car-history>. [Data uzyskania dostępu: 3 Maj 2023].
- [5] Mercedes - Benz, „Omnibus Magazyn - 125 lat autobusów,” Mercedes - Benz, [Online]. Available: https://www.mercedes-benz-bus.com/pl_PL/brand/omnibus-magazin/125-years-buses.html. [Data uzyskania dostępu: 2 Maj 2023].
- [6] T. M. M. i. Miniature, „MMIM Hall of Fame,” [Online]. Available: <https://www.themotormuseuminminiature.co.uk/inv-karl-benz.php>. [Data uzyskania dostępu: 2 5 2023].
- [7] Veritum, „Veritum - rozwiązania informatyczne dla firm,” 10 06 2023. [Online]. Available: <https://www.veritum.pl/>.
- [8] Planbus, „Planbus.pl - program do prowadzenia firmy autobusowej,” 15 06 2023. [Online]. Available: <https://www.planbus.pl/>.
- [9] Optibus, „Optibus - Making Public Transportation Better. Together,” 18 06 2023. [Online]. Available: <https://www.optibus.com/>.
- [10] K. Dmitrowicz-Życkiea, „Transport – wyniki działalności w 2022 r.,” Główny Urząd Statystyczny, Warszawa, 2023.
- [11] A. Adamczyk, „Strategia Zrównoważonego Rozwoju Transportu do 2030 roku,” Rada Ministrów, Warszawa, 2019.
- [12] M. Śmiałek i K. Rybiński, Inżynieria oprogramowania w praktyce. Od wymagań do kodu z językiem UML, Sliwice: Helion, 2023.
- [13] L. Kohnfelder, Po pierwsze: bezpieczeństwo. Przewodnik dla twórców oprogramowania, Gliwice: Helion, 2023.
- [14] J. Albahari i B. Albahari, C# 10. Leksykon kieszonkowy, Gliwice: Helion, 2022.
- [15] J. Skeet, C# od podszewki, Gliwice: Helion, 2020.
- [16] D. Nesteruk, Wzorce projektowe w .NET. Projektowanie zorientowane obiektywne z wykorzystaniem C# i F#, Gliwice: Helion, 2020.
- [17] M. J. Price, C# 10 i .NET 6 dla programistów aplikacji wieloplatformowych, Gliwice: Helion, 2022.
- [18] R. C. Martin, Czysty kod. Podręcznik dobrego programisty, Gliwice: Helion, 2015.
- [19] R. Ye, Projektowanie aplikacji w .NET MAUI. Jak budować doskonałe interfejsy użytkownika dla aplikacji wieloplatformowych, Gliwice: Helion, 2024.