

Termin zajęć <u>11:15</u> Środa	Inżynieria Oprogramowania
Wykonawcy ćwiczenia: 272945 Jakub Barczyński 272924 Jakub Smolarczyk	Grupa nr: 5
Tytuł ćwiczenia: Specyfikacja wymagań funkcjonalnych	Laboratorium nr: 2, 3
Data wykonania ćwiczenia: 16.10.2024	Ocena:
Data oddania ćwiczenia: 23.10.202425	

Spis treści

1 Temat programu: Program obsługujący zakład transportowy	5
1.1 Opis świata rzeczywistego	5
1.2 Opis zasobów ludzkich.	5
1.2.1 Zarządzanie czasem pracy kierowców	5
1.2.2 Zarządzanie załadunkiem i wyładunkiem	5
1.2.3 Monitorowanie pracy przez spedycyjnych	5
1.2.4 Przydzielanie zleceń kierowcom	5
1.2.5 Wsparcie technologiczne	5
1.3 Przepisy i strategia firmy	6
1.4 Dane techniczne	6
1.4.1 Dostęp do systemu:	6
1.4.2 Skalowalność systemu:	6
1.4.3 Placówka firmy	6
1.5 Dane o profilach klientów firmy oraz pracowników firmy	7
1.5.1 Technologia:	7
1.5.2 Bezpieczeństwo:	7
2 Zdefiniowanie wymagań funkcjonalnych i niefunkcjonalnych	7
2.1 Funkcjonalne	7
2.2 Niefunkcjonalne	7
3 Aktorzy	8
4 Diagram przypadków użycia	16
5 Diagramy czynności	17
5.1 Diagram czynności 1 - przypadki użycia:	17
5.1.1 Ogólny schemat diagramu czynności	17
5.1.2 Szczegółowe diagramy czynności	18
5.2 Diagram czynności 2 - przypadki użycia:	19
5.2.1 Ogólny schemat diagramu czynności	19
5.2.2 Szczegółowe diagramy czynności	20
6 Diagram Komponentów	23
7 Warstwa Modelu	24
7.1 Wzorce Projektowe	24
7.2 Kod Warstwa Modelu	24
7.2.1 Zlecenie	24
7.2.2 Użytkownik	26
7.2.3 Spedytor	27
7.2.4 RodzajPowiedzenia	28
7.2.5 PylekPojazd	28
7.2.6 Pylek	29
7.2.7 FabrykaPylkow	29
7.2.8 Powiadomienie	30

7.2.9	Pojazd	31
7.2.10	Kierownik	33
7.2.11	Kierowca	34
7.2.12	IDaneTymczasowe	36
7.2.13	DaneTymczasowe	37
8	Warstwa Prezentera	39
8.0.1	Wzorce Projektowe	39
8.1	Kod Prezenter	40
8.2	PortDoPrezentera	40
8.2.1	AbstrakcyjnaObsluga	41
8.2.2	DAO	41
8.2.3	IPrzydzielanieFunkcjonalnosci	42
8.2.4	KierowcaDAO	42
8.2.5	KierownikDAO	44
8.2.6	PojazdDAO	46
8.2.7	SpedytorDAO	47
8.2.8	StrategiaDAO	49
8.2.9	PowiadomienieDAO	50
8.2.10	ZlecenieDAO	52
8.2.11	KlasaSzablonowaWyboru	54
8.2.12	KreatorListaKierowcow	55
8.2.13	KreatorListaPojazdow	55
8.2.14	Logowanie	55
8.2.15	ObslugaKatalogow	57
8.2.16	ObslugaLadunek	58
8.2.17	ObslugaZlecenia	59
8.2.18	Produkt	59
8.2.19	ProduktListaKierowcow	60
8.2.20	ProduktListaPojazdow	60
8.2.21	PrzydzielanieZlecenia	61
8.2.22	RealizacjaZlecenia	64
8.2.23	SzablonWyboruKierowcy	68
8.2.24	SzablonWyboruPojazdu	69
8.2.25	ZaladunekWyladunek	70
9	Warstwa Widoku	72
9.1	Kod Widok	72
9.1.1	Main	72
9.1.2	IKomunikaty	72
9.1.3	PortDoWidoku	73
9.1.4	Terminal	73
10	Przykładowe dane	76
10.1	Dane Kierowców	76
10.2	Dane Kierowników magazynu	76
10.3	Dane Spedytorzy	76
10.4	Dane Pojazdów	76

10.5 Dane powiadomienia	76
11 Diagram Sekwencji dla logowania	77
12 Złożony diagram sekwencji	78
13 Makietki okienkowej warstwy prezentacji	84
14 Diagram stanów okienkowej warstwy prezentacji	88

1 Temat programu: Program obsługujący zakład transportowy

1.1 Opis świata rzeczywistego

Oprogramowanie ma obejmować takie elementy jak baza danych o dostępności i stanie samochodów, baza nadzorująca zlecenia dostępnych samochodów w firmie, oraz służąca do nadzoru czasu pracy.

1.2 Opis zasobów ludzkich.

1.2.1 Zarządzanie czasem pracy kierowców

Kierowcy mogą prowadzić maksymalnie 9 godzin dziennie, przy czym dwa razy w tygodniu czas ten może zostać wydłużony do 10 godzin. Jednorazowa jazda nie może przekroczyć 4,5 godziny, po upływie 4 godzin system ostrzega kierowcę o konieczności przerwy. przed upływem czasu, kierowca musi udać się na obowiązkową 45-minutową przerwę, którą można podzielić na dwie części: pierwszą trwającą co najmniej 15 minut oraz drugą, która musi trwać minimum 30 minut.

Spedytorzy mają stały wgląd do bieżącego czasu pracy kierowców. W momencie, gdy kierowca przekracza limit, system automatycznie informuje spedytora o konieczności podjęcia działań.

1.2.2 Zarządzanie załadunkiem i wyładunkiem

Załadunek oraz wyładunek są standardowo planowane na 1 godzinę, co wlicza się do czasu pracy kierowcy, chyba że jest to jego ostatni przystanek dnia. Wówczas kierowca po prostu kończy pracę, a rano kontynuuje jazdę z załadowanym lub rozładowanym pojazdem. W przypadku, gdy załadunek lub wyładunek przekraczają przewidziany czas, kierownik magazynu zobowiązany jest zgłosić rzeczywisty czas trwania tych operacji, co jest dokumentowane i wprowadzane do systemu.

1.2.3 Monitorowanie pracy przez spedytorów

Spedytorzy mają dostęp do profili kierowców, które zawierają szczegółowe dane dotyczące ich czasu pracy oraz przestrzegania limitów jazdy bez przewrotu. Program automatycznie informuje o konieczności przerw oraz zakończenia pracy, co pozwala kierowcom na efektywne zarządzanie czasem i realizowanie jak największej liczby zleceń w danym tygodniu.

1.2.4 Przydzielanie zleceń kierowcom

Spedytor oraz kierowca mają dostęp do listy swoich zleceń w bazie. Spedytor ma dodatkowo listę wszystkich pozwalającą na przydział nowych zadań kierowcom. Spedytor posiada również informacje o dostępności pojazdów, dzięki czemu wie ile oraz jakie zlecenia może przydzielić kierowcom.

1.2.5 Wsparcie technologiczne

Program zapewnia bezpośredni dostęp zarówno kierowcom, jak i spedytorom, umożliwiając ciągły nadzór nad czasem pracy, załadunkami oraz wyładunkami. Dzięki automatycznym powiadomieniom, kierowcy mogą efektywnie planować przerwy, a spedytorzy mają pełną kontrolę nad zgodnością operacji z obowiązującymi przepisami prawa, szczególnie z Rozporządzeniem (WE) nr 561/2006 dotyczącym czasu pracy kierowców.

Takie podejście do zarządzania zasobami ludzkimi w firmie transportowej pozwala na zapewnienie bezpieczeństwa na drodze, przestrzeganie przepisów oraz maksymalizację efektywności operacyjnej.

1.3 Przepisy i strategia firmy

Co ogranicza działalność firmy?

1. Przepisy regulujące czas pracy kierowców w Polsce opierają się głównie na Rozporządzeniu (WE) nr 561/2006 Parlamentu Europejskiego i Rady z dnia 15 marca 2006 r.
2. Dzienne limity: Kierowca może prowadzić pojazd maksymalnie przez 9 godzin dziennie. Jednakże, dwa razy w tygodniu, czas ten może zostać przedłużony do 10 godzin.
3. Przerwy w pracy
Po 4,5 godziny nieprzerwanej jazdy, kierowca jest zobowiązany do zrobienia 45-minutowej przerwy, lub dwóch minimum 15 i 30 minutowych przerw.
4. Przerwy te nie mogą być zaliczone do odpoczynku dziennego lub tygodniowego.

1.4 Dane techniczne

1.4.1 Dostęp do systemu:

System musi zapewniać bezpośredni dostęp do programu zarówno dla kierowców, spedycji, jak i kierowników zakładów, z różnymi poziomami uprzywilejowania dostępu:

Kierowcy mają dostęp do swojego profilu po zalogowaniu, mogą sprawdzać czas pracy, listę zleceń do wykonania. Otrzymują również automatyczne powiadomienia o konieczności przerw oraz o limitach czasowych, których muszą przestrzegać.

Spedytorzy mają szerszy dostęp, obejmujący monitoring bieżącego czasu pracy wszystkich kierowców przypisanych do ich obszaru odpowiedzialności. Spedytorzy otrzymują powiadomienia o przekroczeniach limitów oraz mogą przeglądać dane dotyczące terminowości realizowanych zleceń oraz czasu załadunku w danych magazynach, dzięki czemu mogą optymalizować rozdzielenie pracy.

Kierownicy magazynów po zalogowaniu muszą poinformować spedytora o przedłużonym załadunku lub wyładunku, jeżeli takowy miał miejsce.

1.4.2 Skalowalność systemu:

System musi obsługiwać kilkadziesiąt aktywnych profili kierowców przypisanych do jednego zakładu jednocześnie. Z tego względu architektura systemu musi być skalowalna i elastyczna, umożliwiając obsługę dziesiątek użytkowników jednocześnie, przy jednoczesnym zachowaniu wysokiej wydajności i bezpieczeństwa danych. Program przewiduje obsługę:

1. do 100 kierowców (razem we wszystkich placówkach)
2. do 1000 zleceń
3. do 10 placówek

1.4.3 Placówka firmy

Główna placówka firmy znajduje się w Kątach Wrocławskich

1.5 Dane o profilach klientów firmy oraz pracowników firmy

Aplikacja jest zaprojektowana do użytku wewnętrznego, bez dostępu klientów do niej. Zlecenia pozywane są z bazy danych.

1.5.1 Technologia:

Zaleca się, aby system był oparty na technologii Java, z uwagi na jej wydajność, stabilność oraz szerokie wsparcie dla budowy systemów rozproszonych. Java oferuje odpowiednie narzędzia do tworzenia skalowalnych i wielowarstwowych aplikacji, które mogą działać w środowiskach rozproszonych, co jest kluczowe w przypadku firmy o strukturze obejmującej kilka zakładów. Ponadto, Java wspiera rozwój aplikacji webowych, co umożliwia łatwy dostęp do systemu zarówno z komputerów, jak i urządzeń mobilnych.

1.5.2 Bezpieczeństwo:

System musi gwarantować odpowiedni poziom bezpieczeństwa danych, zarówno w zakresie autoryzacji dostępu, jak i ochrony danych osobowych kierowców oraz informacji związanych z pracą. Konieczne jest zastosowanie mechanizmów szyfrowania haseł.

2 Zdefiniowanie wymagań funkcjonalnych i niefunkcjonalnych

2.1 Funkcjonalne

1. System automatycznie informuje Spedytora o przekroczeniu norm czasu pracy przez kierowców.
2. System monitoruje czas pracy kierowców, uwzględniając obowiązujące limity czasowe.
3. Spedytor ma dostęp do listy dostępnych pojazdów oraz ich stanu technicznego.
4. System informuje kierowcę o konieczności zrobienia przerwy oraz o czasie, w którym może kontynuować pracę.
5. System przesyła Spedytorowi informacje od kierownika magazynu o nadmiarowym czasie załadunku.
6. Spedytor ustala listy zleceń na podstawie zapotrzebowania na usługi firmy oraz dostępności floty pojazdów.
7. Kierowca ma dostęp do swojej listy zleceń.
8. Spedytor widzi pełną listę zleceń i ich stopień realizacji.
9. System umożliwia Spedytorowi wybór w liczbę przekroczeń limitów czasowych przez kierowców w danym miesiącu oraz w dni, w których kierowcy zrealizowali wszystkie przypisane im zadania.

2.2 Niefunkcjonalne

1. Dostęp do systemu mają wszyscy pracownicy firmy, w tym kierowcy, kierownicy oraz Spedytor.
2. Zakres dostępnych funkcji systemu jest ograniczony do uprawnień przypisanych do stanowiska (Spedytor, kierownik lub kierowca).

- System jest zabezpieczony procesem logowania, wymagającym unikalnych danych uwierzytelniających użytkownika.

3 Aktorzy

Aktor	Opis	Przypadki użycia
Kierowca	Kierowca odpowiada za prowadzenie pojazdu, przestrzeganie limitów czasu pracy oraz raportowanie czasu załadunku i wyładunku.	<ul style="list-style-type: none"> 1: Monitorowanie limitów czasu pracy kierowcy. Kierowca ma stały wgląd w to, czy mieści się limitach 5: Przeglądanie listy zleceń. Kierowca posiada stały wgląd w swoje aktualne zlecenia 8: Przeglądanie profilu 7 Powiadomienie o przekroczeniu norm czasu pracy Powiązane przez relacje extend z PU1. Kierowca zostaje powiadomiony, o przekroczeniu limitu jazdy bez przerwy. 10 Zgłoszenie dotarcia do celu. Powiązanie przez relację extend z PU1. Kierowca zgłasza dotarcie do celu.
Kierownik magazynu	Odpowiada za nadzór nad procesem załadunku i wyładunku towarów, a także potwierdzanie faktycznego czasu trwania tych operacji.	<ul style="list-style-type: none"> 2: Nadzór nad czasem załadunku i wyładunku. 8: Przeglądanie profilu. Kierownik ma stały wgląd w swój profil. 9: Wprowadzenie informacji o czasie załadunku. powiązanie przez extend z PU2. W sytuacji przekroczenia czasu podstawowego kierownik magazynu zobowiązany jest wprowadzić faktyczny czas do systemu.

Aktor	Opis	Przypadki użycia
Spedytor	Spedytor zarządza kierowcami oraz przydzielaniem im zleceń, monitoruje przestrzeganie norm czasu pracy oraz stan techniczny pojazdów.	<ul style="list-style-type: none"> • 1: Monitorowanie limitów czasu pracy kierowcy. Po przez relacje extend PU1. Spedytor ma stały wgląd w limity czasowe kierowcy • 2: Nadzór nad czasem załadunku i wyładunku. Powiązanie poprzez relacje extend PU1. Spedytor ma dostęp do informacji o czasie załadunku. • 3: Przydzielanie zleceń kierowcom. Powiązanie poprzez relacje include z PU4 • 4: Wgląd w listę dostępnych pojazdów i ich stan. Spedytor posiada informacje o stanie wszystkich aut w firmie. • 5: Przeglądanie listy zleceń. Spedytor otrzymuje stały dostęp do informacji o zleceniach z bazy danych. • 6: Nadzór nad przestrzeganiem limitów pracy przez kierowców. Spedytor otrzymuje informacje o danej sytuacji z ograniczeniami dotyczącymi kierowców. • 7: Powiadomienia o przekroczeniu norm czasu pracy. Spedytor otrzymuje powiadomienia o przekroczeniu norm czasowych przez kierowców. • 8: Przeglądanie profilu. • 9: Wprowadzenie informacji o czasie załadunku. Spedytor otrzymuje informacje o czasie załadunku w magazynie. • 10: Zgłoszenie dotarcia do celu. Spedytor otrzymuje informacje, o tym że kierowca dotarł do celu.

Aktor	Opis	Przypadki użycia
Baza danych	Baza danych przechowuje informacje o kierowcach, zleceniach, pojazdach oraz raporty o przestrzeganiu norm czasu pracy.	<ul style="list-style-type: none"> • 3: Zapewnia wgląd w aktualne limity czasu pracy kierowców. • 4: Zapewnia wgląd w listę dostępnych pojazdów i ich stan. • 5: Umożliwia przeglądanie listy zleceń, • 6: Nadzór nad przestrzeganiem limitów pracy przez kierowców. Dostarcza informacje o aktualnej sytuacji kierowców oraz czy przestrzegają ograniczeń. • 8 Umożliwia wgląd w listę dostępnych pojazdów i ich stan. Posiada informacje o stanie pojazdów.

Przypadki użycia

1. Monitorowanie limitów czasu pracy kierowcy

CEL: Monitorowanie czasu pracy kierowcy oraz ostrzeganie go przed przekroczeniem dopuszczalnych limitów pracy.

WS (warunki wstępne): Inicjalizacja przy uruchomieniu programu.

WK (warunki końcowe): Ostrzeżenie o zbliżającym się przekroczeniu limitu czasu pracy jest wysyłane, a kierowca musi zrobić przerwę zgodnie z przepisami.

PRZEBIEG:

1. Kierowca loguje się do systemu.
2. Kierowca rozpoczyna jazdę po otrzymaniu zlecenia.
3. System rejestruje rozpoczęcie pracy kierowcy i monitoruje czas jazdy.
4. Po 4 godzinach jazdy system wysyła ostrzeżenie do kierowcy, informując o konieczności zrobienia przerwy PU6.
5. Po 4,5 godziny jazdy system informuje kierowcę, że musi się zatrzymać na przerwę trwającą 45 minut (lub dzieloną na dwie części – 15 i 30 minut).
6. Wywołanie **PU7 Powiadomienie o przekroczeniu norm czasu pracy** w przypadku gdy kierowca przekroczy limit 4,5 godziny jazdy.
7. Kierowca udaje się na przerwę, po której powtarzane są kroki PU1 począwszy od trzeciego lub wywołuje **PU10 Zgłoszenie dotarcia do celu**, w celu rozpoczęcia wyładunku.

2. Nadzór nad czasem załadunku i wyładunku

CEL: Monitorowanie czasu załadunku i wyładunku towarów, aby upewnić się, że proces nie przekracza ustalonego limitu.

WS (warunki wstępne): Kierowca musi zgłosić przyjazd do miejsca załadunku/wyładunku (PU10 Zgłoszenie dotarcia do celu).

WK (warunki końcowe): Jeśli załadunek/wyładunek przekroczy 1 godzinę, spedycja zostaje poinformowana o opóźnieniu.

PRZEBIEG:

1. Kierowca zgłasza przyjazd do miejsca załadunku/wyładunku poprzez system z poziomu **PU10 Zgłoszenie dotarcia do celu**.
2. Kierownik loguje się do systemu w celu rozpoczęcia załadunku/wyładunku.
3. System rozpoczyna monitorowanie czasu trwania załadunku/wyładunku.
4. Po upływie 1 godziny, jeśli proces nie został zakończony, kierownik magazynu wywołuje **PU9 Wprowadzenie informacji o czasie załadunku**.
5. System informuje spedycji o przekroczeniu czasu załadunku/wyładunku wprowadzonym przez kierownika magazynu i uwzględnia opóźnienie w limicie czasu pracy kierowcy.
6. W przypadku, gdy załadunek/wyładunek odbywa się na ostatnim przystanku dnia, czas ten nie jest wliczany do czasu pracy kierowcy.

3. Przydzielanie zleceń kierowcom

CEL: Przypisywanie zleceń do kierowców na podstawie ich dostępności oraz limitów czasu pracy.

WS (warunki wstępne): Inicjalizacja przy uruchomieniu programu.

WK (warunki końcowe): Wybrany kierowca otrzymuje odpowiednie zlecenie, a system zapewnia, że czas pracy kierowcy nie przekroczy limitów.

PRZEBIEG:

1. Spedytor loguje się do systemu.
2. Spedytor wywołuje **PU4 Wgląd w listę dostępnych pojazdów i ich stan**, wybierając odpowiedni pojazd do zlecenia.
3. Spedytor wybiera kierowcę do zlecenia.
4. Na podstawie dostępnych danych, spedytor tworzy listę zleceń dla kierowców.
5. Spedytor przypisuje zlecenia do odpowiednich kierowców, biorąc pod uwagę ich czas pracy oraz aktualne limity, na podstawie pochodzących z bazy danych.
6. System upewnia spedytora, czy przypisane zlecenia są wykonalne w ramach pozostałego czasu pracy kierowcy.

4. Wgląd w listę dostępnych pojazdów i ich stan

CEL: Umożliwienie spedytorowi przeglądania listy dostępnych pojazdów oraz ich stanu technicznego.

WS (warunki wstępne): Inicjalizacja przy uruchomieniu programu lub może być wywołany z PU3 Przydzielanie zleceń kierowcom.

WK (warunki końcowe): Spedytor wybiera pojazd i przypisuje go do kierowcy na podstawie stanu technicznego pojazdu i dostępności.

PRZEBIEG:

1. Spedytor loguje się do systemu, chyba że PU jest wywołany z poziomu **PU3 Przydzielanie zleceń kierowcom**.
2. System wyświetla aktualną listę dostępnych pojazdów wraz z informacjami o ich stanie technicznym oraz czy dany pojazd wymaga naprawy.
3. Spedytor przegląda dane i wybiera odpowiedni pojazd.

5. Przeglądanie listy zleceń

CEL: Umożliwienie użytkownikowi przeglądania dostępnych dla niego listy zleceń. Dla kierowcy jest to jego własna lista, dla spedytora są to listy kierowców.

WS (warunki wstępne): Inicjalizacja przy uruchomieniu programu.

WK (warunki końcowe): Użytkownik przegląda listę zleceń oraz szczegóły dotyczące załadunku/wyładunku i limitów czasowych.

PRZEBIEG (kierowca):

1. Kierowca loguje się do systemu.
2. Kierowca wybiera opcję "Zlecenia".
3. System wyświetla listę zleceń przypisanych do danego kierowcy.
4. Kierowca przegląda szczegóły zleceń, w tym adresy, czas załadunku/wyładunku oraz limity czasowe.
5. Kierowca podejmuje działania zgodnie z wymaganiami zlecenia.

PRZEBIEG (spedytor):

1. Spedytor loguje się do systemu.
2. Spedytor wybiera opcję "Zlecenia".
3. Spedytor wybiera danego kierowcę.
4. System wyświetla listę zleceń przypisanych do wybranego kierowcy.
5. Spedytor przegląda szczegóły zleceń, w tym adresy, czas załadunku/wyładunku oraz limity czasowe.

6. Nadzór nad przestrzeganiem limitów pracy przez kierowców

CEL: Zapewnienie, że kierowcy przestrzegają limitów czasu pracy, a spedytor może monitorować te informacje w formie raportów.

WS (warunki wstępne): Inicjalizacja przy uruchomieniu programu.

WK (warunki końcowe): Spedytor otrzymuje dostęp do raportów o liczbie przekroczeń limitów czasu pracy dla każdego kierowcy.

PRZEBIEG:

1. Spedytor loguje się do systemu.
2. Program sumuje wszystkie upomnienia każdego z kierowców.
3. System pierwszego dnia miesiąca generuje raporty o liczbie przekroczonych limitów czasu pracy przez każdego kierowcę w ostatnim miesiącu.
4. Program wysyła raporty każdego z kierowców do spedytora.
5. Na podstawie raportów spedytor podejmuje decyzje dotyczące planowania przyszłych zleceń.

7. Powiadomienia o przekroczeniu norm czasu pracy

CEL: Automatyczne informowanie spedytora o przekroczeniu przez kierowców ustalonych limitów czasu pracy.

WS (warunki wstępne): Może być wywołany z PU1 Monitorowanie limitów czasu pracy kierowcy.

WK (warunki końcowe): Spedytor otrzymuje automatyczne powiadomienie o przekroczeniu norm czasu pracy przez kierowcę.

PRZEBIEG:

1. Kierowca z poziomu **PU1 Monitorowanie limitów czasu pracy** kierowcy przekracza ustalony limit czasu pracy.
2. System automatycznie generuje powiadomienie o przekroczeniu norm czasu pracy dla spedytora.
3. Spedytor otrzymuje powiadomienie o przekroczeniu limitu czasu pracy przez kierowcę.
4. System zlicza liczbę upomnień w każdym miesiącu. Gdy liczba upomnień przekracza 5, na kierowcę nakładana jest kara pieniężna.

ALTERNatywny PRZEBIEG:

1. Kierowca z poziomu **PU1 Monitorowanie limitów czasu pracy** kierowcy przekracza ustalony limit czasu pracy.

2. System automatycznie generuje powiadomienie o przekroczeniu norm czasu pracy dla spedytora.
3. Spedytor otrzymuje powiadomienie o przekroczeniu limitu czasu pracy przez kierowcę.
4. System zlicza liczbę upomnień w każdym miesiącu. Gdy liczba upomnień jest mniejsza lub równa 5, kierowca nie otrzymuje kary pieniężnej.

8. Przeglądanie profilu

CEL: Przegladanie profilu swojego lub innych w zależności od uprawnień

WS (warunki wstępne): Inicjalizacja przy uruchomieniu programu.

WK (warunki końcowe): Użytkownik ma możliwość przeglądania i edytowania swoich zgodnie z przydzielonymi uprawnieniami.

PRZEBIEG:

1. Użytkownik loguje się do systemu.
2. Użytkownik wybiera opcję "Profil".
3. System wyświetla dane użytkownika, w tym: imię, nazwisko, stanowisko oraz (dla kierowców) raporty o przestrzeganiu limitów czasu pracy. Spedytor ponadto może przeglądać wszystkich kierowników magazynów oraz kierowców.

ALTERNatywny PRZEBIEG:

1. Użytkownik loguje się do systemu.
2. Użytkownik wybiera opcję "Profil".
3. System wyświetla dane użytkownika, w tym: imię, nazwisko, stanowisko oraz (dla kierowców) raporty o przestrzeganiu limitów czasu pracy. Spedytor ponadto może przeglądać wszystkich kierowników magazynów oraz kierowców.
4. Użytkownik edytuje wybrane dane (w zależności od uprawnień).
5. Po dokonaniu zmian, użytkownik zapisuje zmodyfikowane dane.
6. System aktualizuje informacje w bazie danych i potwierdza dokonanie zmian.

9. Wprowadzanie informacji o czasie załadunku

CEL: Poinformowanie spedytora o niestandardowym czasie trwania załadunku/wyładunku.

WS (warunki wstępne): Może być wywołany z PU2 Nadzór nad czasem załadunku i wyładunku, jeśli standardowy czas został przekroczyony.

WK (warunki końcowe): Z pomocą systemu spedytor posiada informację o przekroczonym czasie.

PRZEBIEG:

1. Z poziomu **PU2 Nadzór nad czasem załadunku i wyładunku** przekroczony zostaje założony czas załadunku/wyładunku.
2. Kierownik wprowadza do systemu niestandardowy czas załadunku/wyładunku.
3. System informuje spedytora o niestandardowym czasie danego załadunku lub wyładunku.

10. Zgłoszenie dotarcia do celu

CEL: Kierowca kończy dany przejazd.

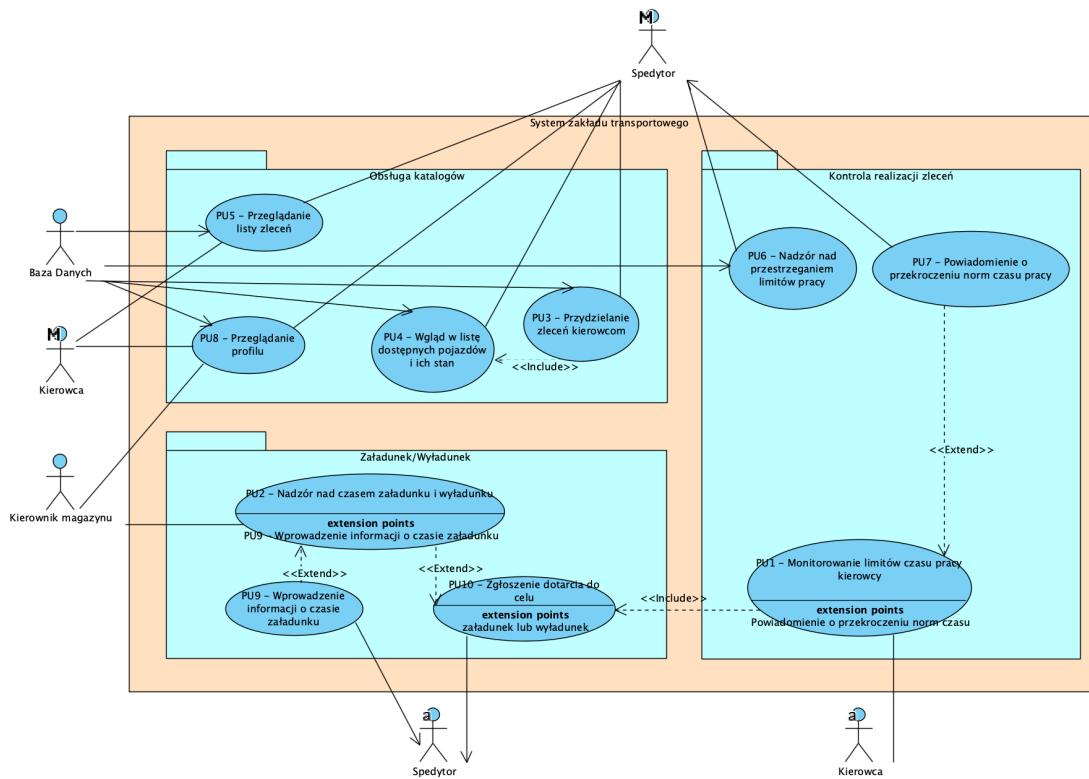
WS (warunki wstępne): Może być wywołany z PU1 Monitorowanie limitów czasu pracy kierowcy.

WK (warunki końcowe): Poinformowanie spedytora o osiągnięciu celu lub wywołanie PU2 Nadzór nad czasem załadunku i wyładunku, jeśli dany przystanek nie jest dla kierowcy ostatnim w ciągu dnia.

PRZEBIEG:

1. Z poziomu **PU1 Monitorowanie limitów czasu pracy kierowcy** kierowca dociera do celu.
2. Kierowca wysyła do systemu informację o dotarciu do celu.
3. Spedytor otrzymuje informację o zakończeniu danego przejazdu.
4. System resetuje czas jazdy kierowcy.
5. Wywołane **PU2 Nadzór nad czasem załadunku i wyładunku**, jeśli kierowca ma jeszcze do wykonania zlecenia danego dnia.

4 Diagram przypadków użycia

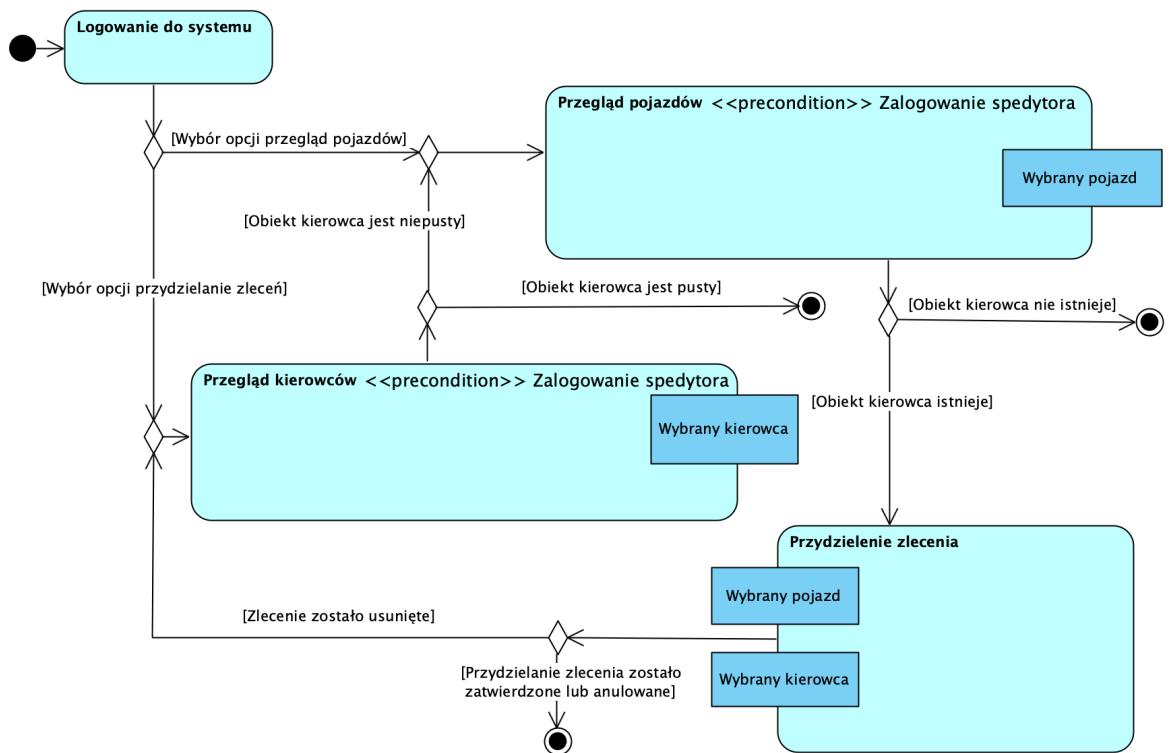


5 Diagramy czynności

5.1 Diagram czynności 1 - przypadki użycia:

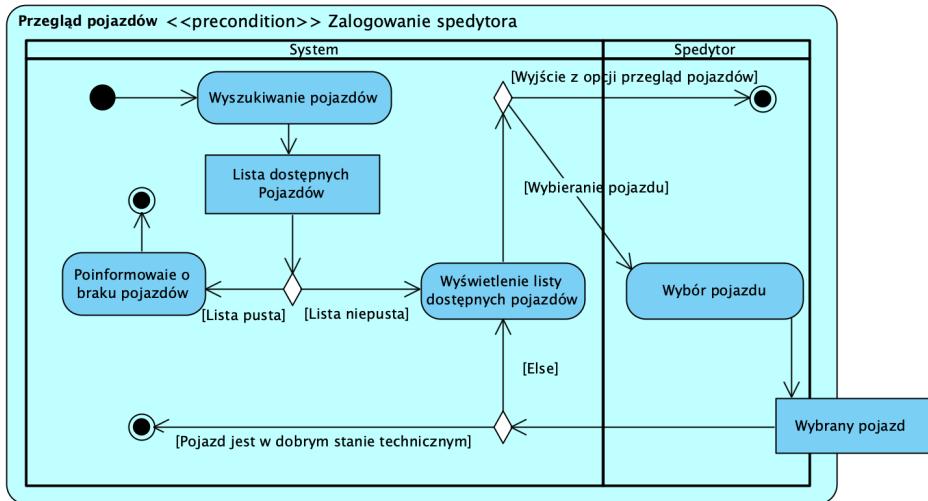
1. PU3 - Przydzielanie zleceń kierowcom
2. PU4 - Wgląd w listę pojazdów i ich stan

5.1.1 Ogólny schemat diagramu czynności

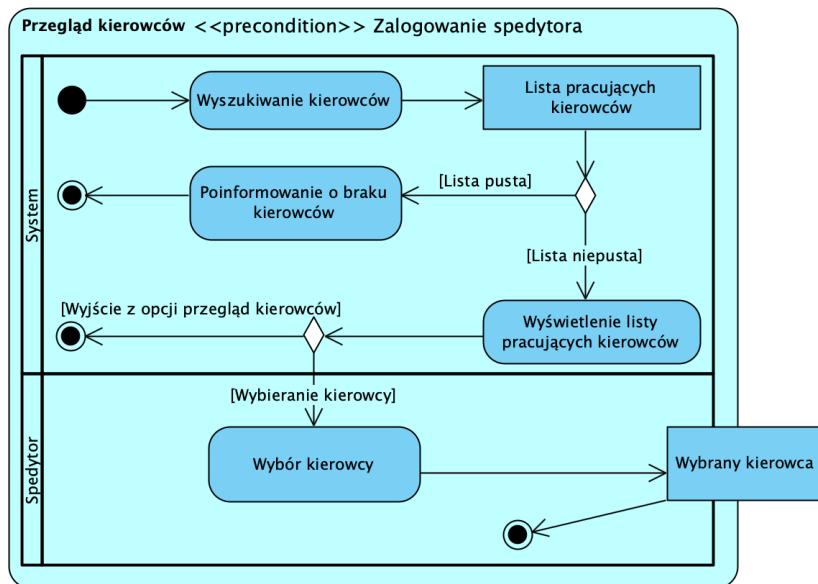


Rysunek 1: Ogólny schemat pierwszego diagramu czynności

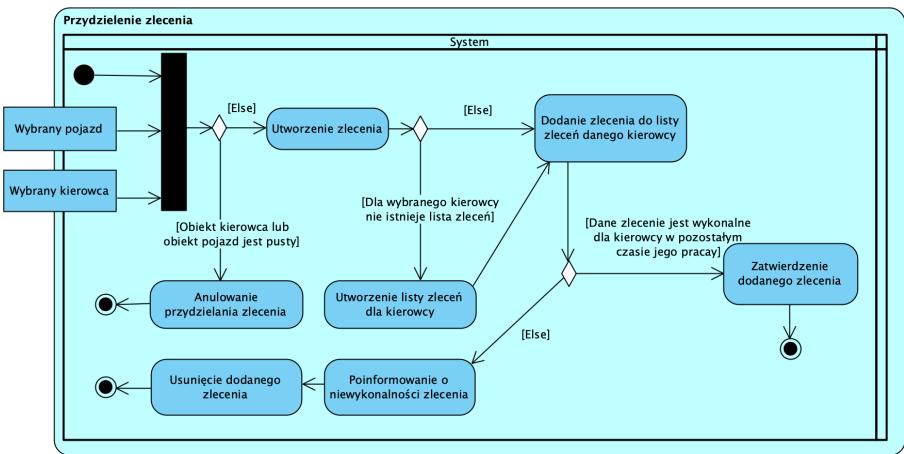
5.1.2 Szczegółowe diagramy czynności



Rysunek 2: Czynność <przegląd pojazdów>



Rysunek 3: Czynność <przegląd kierowców>

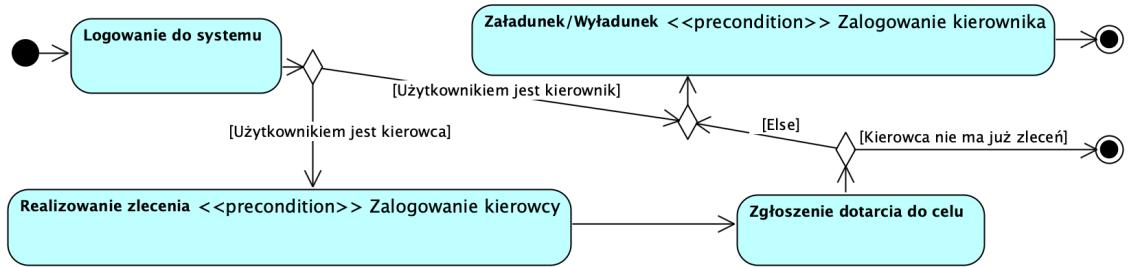


Rysunek 4: Czynność <przydzielenie zleceń kierowcom>

5.2 Diagram czynności 2 - przypadki użycia:

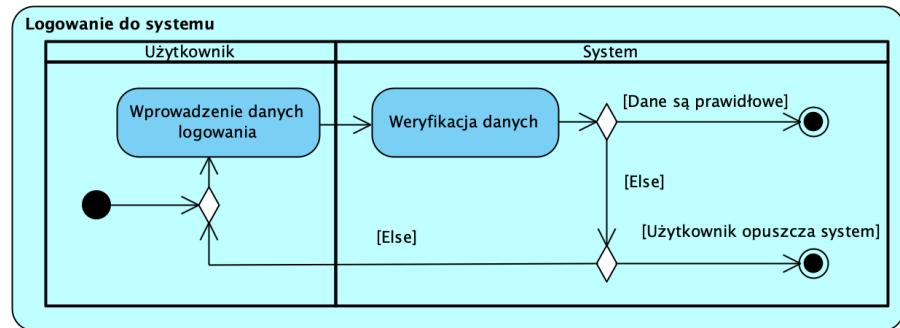
1. PU1 - Monitorowanie limitów czasu pracy kierowcy
2. PU2 - Nadzór nad czasem załadunku i wyładunku
3. PU7 - Powiadomienie o przekroczeniu norm czasu pracy
4. PU9 - Wprowadzenie informacji o czasie załadunku
5. PU10 - Zgłoszenie dotarcia do celu

5.2.1 Ogólny schemat diagramu czynności

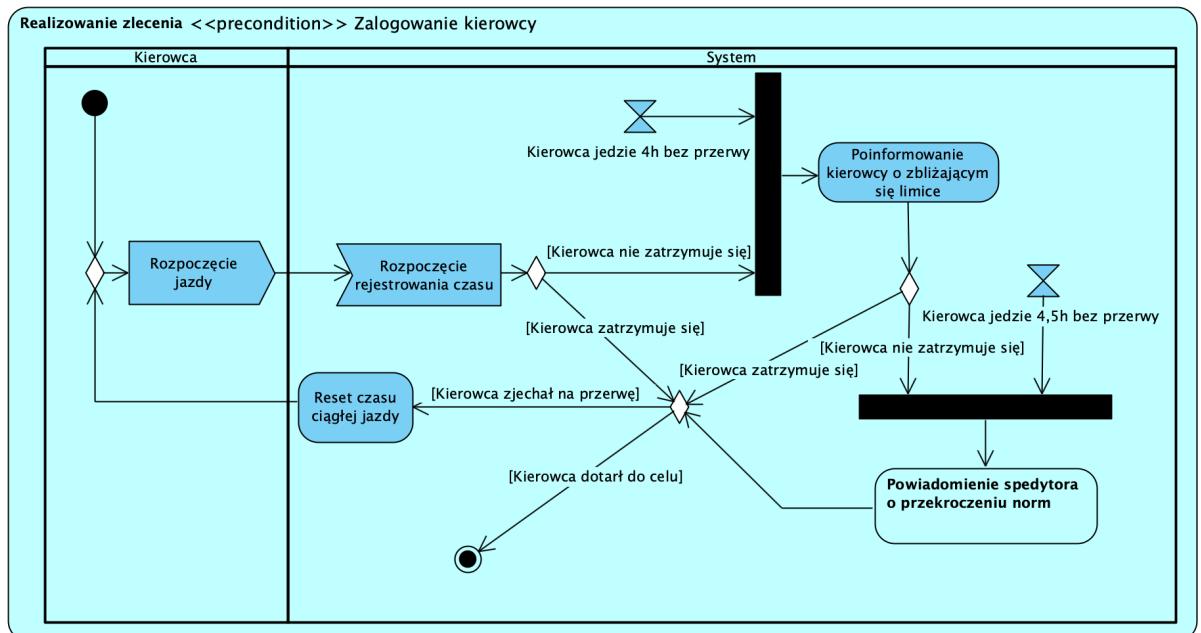


Rysunek 5: Ogólny schemat drugiego diagramu czynności

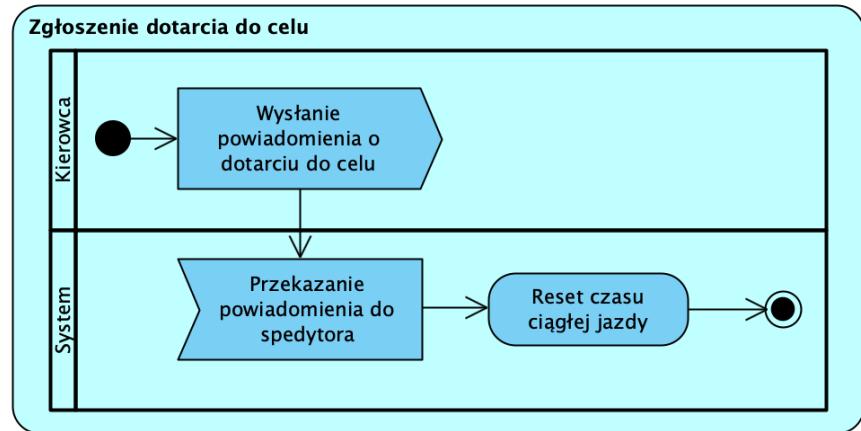
5.2.2 Szczegółowe diagramy czynności



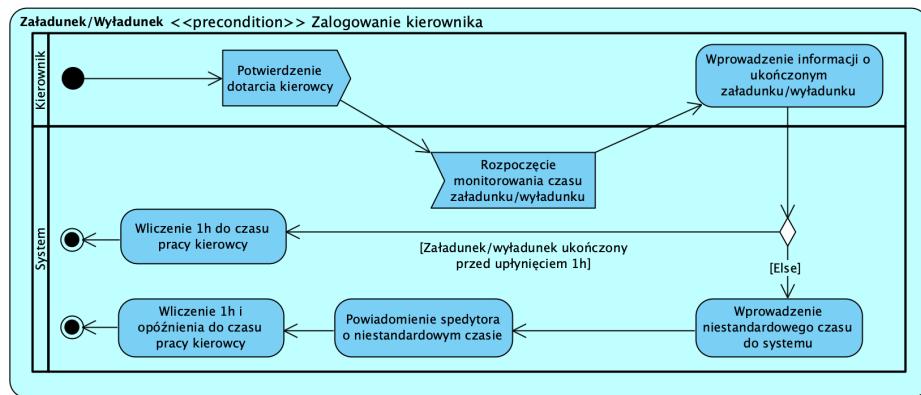
Rysunek 6: Czynność <logowanie do systemu>



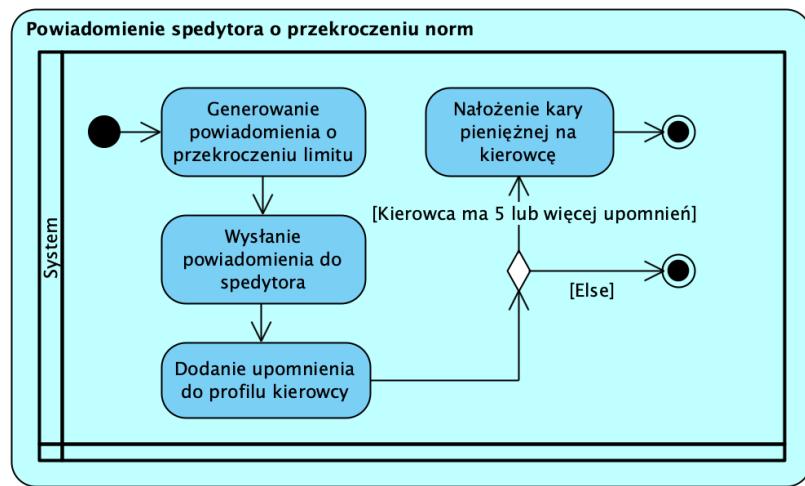
Rysunek 7: Czynność <Monitorowanie limitów czasu jazdy kierowcy>



Rysunek 8: Czynność <Zgłoszenie dotarcia do celu>

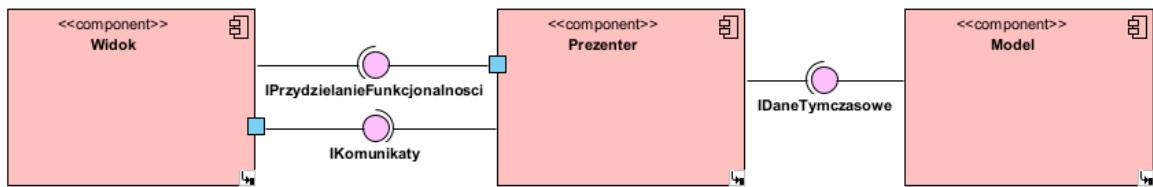


Rysunek 9: Czynność <Nadzór nad Załadunkiem pojazdu>



Rysunek 10: Czynność <Powiadamianie spedytora o przekroczeniu norm>

6 Diagram Komponentów



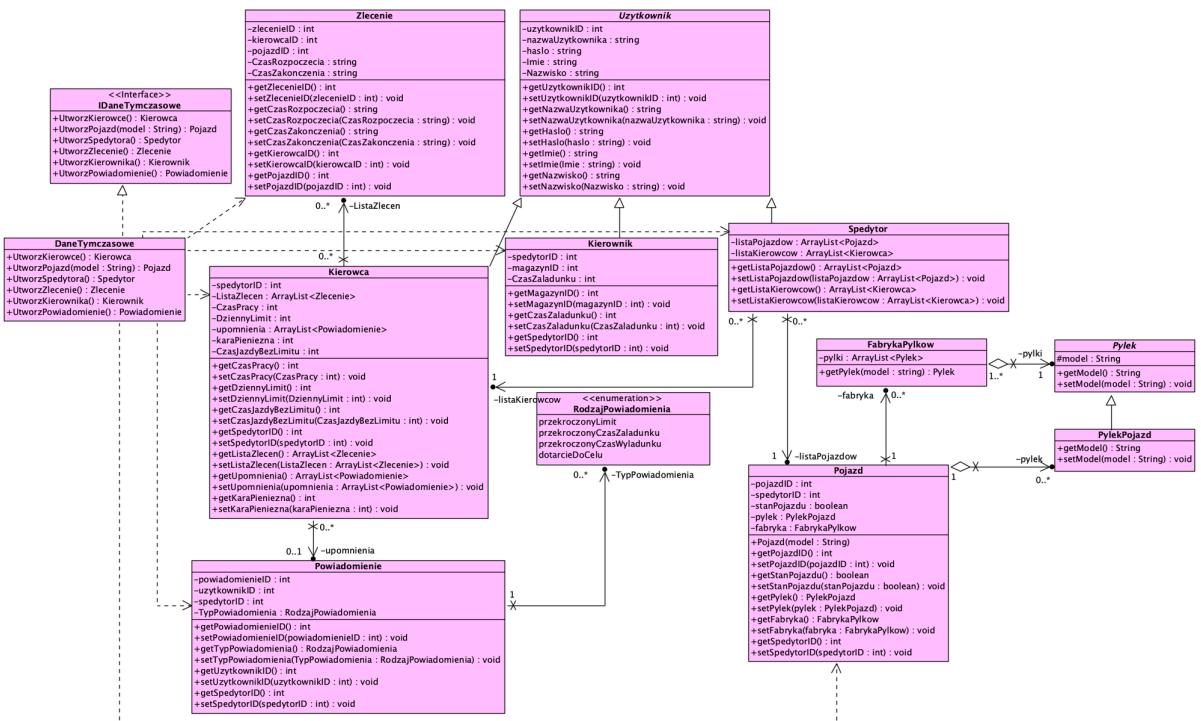
1. Widok: Komponent odpowiedzialny za interakcję z użytkownikiem. Wyświetla dane przekazane przez prezentera i wysyła akcje użytkownika do prezentera. Jest połączony z warstwą prezentera za pomocą interfejsu `IPrzydzielanieFunkcjonalnosci`, który udostępnia metody umożliwiające interakcję z logiką aplikacji.
2. Prezenter: Centralny komponent, który łączy widok i model. Odbiera dane od widoku, przetwarza je, i komunikuje się z modelem w celu wykonania logiki biznesowej. Wysyła przetworzone dane z powrotem do widoku. Połączony z: Widokiem za pomocą interfejsu `IPrzydzielanieFunkcjonalnosci`, dzięki czemu może zarządzać funkcjami dostępnymi dla widoku. Modelem za pomocą interfejsu `IDaneTymczasowe`, co umożliwia odczyt i zapis danych w modelu.
3. Model: Komponent odpowiedzialny za logikę biznesową i zarządzanie danymi. Przechowuje dane i udostępnia je pozostałym komponentom. Może wykonywać złożone operacje na danych, których wyniki są zwracane prezenterowi.
4. Interfejsy:
 - `IPrzydzielanieFunkcjonalnosci`: Interfejs między widokiem a prezenterem. Udostępnia metody, które widok może wywoływać, np. przekazanie żądań użytkownika do prezentera.
 - `IDaneTymczasowe`: Interfejs między prezenterem a modelem.

7 Warstwa Modelu

Warstwa modelu odpowiada za logikę biznesową oraz zarządzanie danymi aplikacji. Przechowuje dane, odpowiada za ich przetwarzanie i udostępnianie ich w odpowiednim formacie.

7.1 Wzorce Projektowe

- Pylek:** Optymalizuje zużycie pamięci poprzez współdzielenie wspólnych danych (np. fabryka pojazdów).



Rysunek 11: Diagram klas komponentu Model

7.2 Kod Warstwa Modelu

7.2.1 Zlecenie

```
package Model;

public class Zlecenie {

    private int zlecenieID;
    private int kierowcaID;
    private int pojazdID;
```

```

private String CzasRozpoczecia;
private String CzasZakonczenia;

public int getZlecenieID() {
    return this.zlecenieID;
}

/**
 *
 * @param zlecenieID
 */
public void setZlecenieID(int zlecenieID) {
    this.zlecenieID = zlecenieID;
}

public String getCzasRozpoczecia() {
    return CzasRozpoczecia;
}

/**
 *
 * @param CzasRozpoczecia
 */
public void setCzasRozpoczecia(String CzasRozpoczecia) {
    this.CzasZakonczenia = CzasRozpoczecia;
}

public String getCzasZakonczenia() {
    return CzasZakonczenia;
}

/**
 *
 * @param CzasZakonczenia
 */
public void setCzasZakonczenia(String CzasZakonczenia) {
    this.CzasZakonczenia = CzasZakonczenia;
}

public int getKierowcaID() {
    return this.kierowcaID;
}

/**
 *
 * @param kierowcaID
 */
public void setKierowcaID(int kierowcaID) {

```

```

        this.kierowcaID = kierowcaID;
    }

    public int getPojazdID() {
        return this.pojazdID;
    }

    /**
     *
     * @param pojazdID
     */
    public void setPojazdID(int pojazdID) {
        this.pojazdID = pojazdID;
    }

}

```

7.2.2 Uzytkownik

```

package Model;

public abstract class Uzytkownik {

    private int uzytkownikID;
    private String nazwaUzytkownika;
    private String haslo;
    private String Imie;
    private String Nazwisko;

    public int getUzytkownikID() {
        return this.uzytkownikID;
    }

    /**
     *
     * @param uzytkownikID
     */
    public void setUzytkownikID(int uzytkownikID) {
        this.uzytkownikID = uzytkownikID;
    }

    public String getNazwaUzytkownika() {
        return this.nazwaUzytkownika;
    }

    /**
     *
     */

```

```

    * @param nazwaUzytkownika
    */
public void setNazwaUzytkownika(String nazwaUzytkownika) {
    this.nazwaUzytkownika = nazwaUzytkownika;
}

public String getHaslo() {
    return this.haslo;
}

/**
 *
 * @param haslo
 */
public void setHaslo(String haslo) {
    this.haslo = haslo;
}

public String getImie() {
    return Imie;
}

/**
 *
 * @param Imie
 */
public void setImie(String Imie) {
    this.Imie = Imie;
}

public String getNazwisko() {
    return Nazwisko;
}

/**
 *
 * @param Nazwisko
 */
public void setNazwisko(String Nazwisko) {
    this.Nazwisko = Nazwisko;
}

}

```

7.2.3 Spedytor

```
package Model;
```

```

import java.util.ArrayList;

public class Spedytor extends Uzytkownik {

    private ArrayList<Pojazd> listaPojazdow;
    private ArrayList<Kierowca> listaKierowcow;

    public ArrayList<Pojazd> getListaPojazdow() {
        return this.listaPojazdow;
    }

    /**
     *
     * @param listaPojazdow
     */
    public void setListaPojazdow(ArrayList<Pojazd> listaPojazdow) {
        this.listaPojazdow = listaPojazdow;
    }

    public ArrayList<Kierowca> getListaKierowcow() {
        return this.listaKierowcow;
    }

    /**
     *
     * @param listaKierowcow
     */
    public void setListaKierowcow(ArrayList<Kierowca> listaKierowcow) {
        this.listaKierowcow = listaKierowcow;
    }
}

```

7.2.4 RodzajPowiadomienia

```

package Model;

public enum RodzajPowiadomienia {
    przekroczonyLimit,
    przekroczonyCzasZaladunku,
    przekroczonyCzasWyladunku,
    dotarcieDoCelu
}

```

7.2.5 PylekPojazd

```

package Model;

public class PylekPojazd extends Pylek {

    public String getModel() {
        return model;
    }

    /**
     *
     * @param model
     */
    public void setModel(String model) {
        this.model = model;
    }

}

```

7.2.6 Pylek

```

package Model;

public abstract class Pylek {

    protected String model;

    public String getModel() {
        return this.model;
    }

    /**
     *
     * @param model
     */
    public void setModel(String model) {
        this.model = model;
    }

}

```

7.2.7 FabrykaPylkow

```

package Model;

import java.util.ArrayList;

public class FabrykaPylkow {

```

```

private ArrayList<Pylek> pylki = new ArrayList<>();

/**
 *
 * @param model
 */
public Pylek getPylek(String model) {
    int index = 0;
    for(Pylek element : pylki){
        if(element.getModel().equals(model)){
            return element;
        }
        index++;
    }
    Pylek pylek = new PylekPojazd();
    pylek.setModel(model);
    return pylek;
}

}

```

7.2.8 Powiadomienie

```

package Model;

public class Powiadomienie {

    private int powiadomienieID;
    private int uzytkownikID;
    private int spedycytorID;
    private RodzajPowiadomienia TypPowiadomienia;

    public int getPowiadomienieID() {
        return this.powiadomienieID;
    }

    /**
     *
     * @param powiadomienieID
     */
    public void setPowiadomienieID(int powiadomienieID) {
        this.powiadomienieID = powiadomienieID;
    }

    public RodzajPowiadomienia getTypPowiadomienia() {
        return TypPowiadomienia;
    }
}

```

```

    }

    /**
     *
     * @param TypPowiadomienia
     */
    public void setTypPowiadomienia(RodzajPowiedomienia TypPowiadomienia) {
        this.TypPowiadomienia = TypPowiadomienia;
    }

    public int getUzytkownikID() {
        return this.uzytkownikID;
    }

    /**
     *
     * @param uzytkownikID
     */
    public void setUzytkownikID(int uzytkownikID) {
        this.uzytkownikID = uzytkownikID;
    }

    public int getSpedytorID() {
        return this.spedytorID;
    }

    /**
     *
     * @param spedytorID
     */
    public void setSpedytorID(int spedytorID) {
        this.spedytorID = spedytorID;
    }
}

```

7.2.9 Pojazd

```

package Model;

import java.util.*;

public class Pojazd {

    private int pojazdID;
    private int spedytorID;
    private boolean stanPojazdu;

```

```

private PylekPojazd pylek;
private FabrykaPylkow fabryka = new FabrykaPylkow();

/**
 *
 * @param model
 */
public Pojazd(String model) {
    this.pylek = (PylekPojazd)fabryka.getPylek(model);
}

public int getPojazdID() {
    return this.pojazdID;
}

/**
 *
 * @param pojazdID
 */
public void setPojazdID(int pojazdID) {
    this.pojazdID = pojazdID;
}

public boolean getStanPojazdu() {
    return this.stanPojazdu;
}

/**
 *
 * @param stanPojazdu
 */
public void setStanPojazdu(boolean stanPojazdu) {
    this.stanPojazdu = stanPojazdu;
}

public PylekPojazd getPylek() {
    return this.pylek;
}

/**
 *
 * @param pylek
 */
public void setPylek(PylekPojazd pylek) {
    this.pylek = pylek;
}

public FabrykaPylkow getFabryka() {

```

```

        return this.fabryka;
    }

    /**
     *
     * @param fabryka
     */
    public void setFabryka(FabrykaPylkow fabryka) {
        this.fabryka = fabryka;
    }

    public int getSpedytorID() {
        return this.spedytorID;
    }

    /**
     *
     * @param spedytorID
     */
    public void setSpedytorID(int spedytorID) {
        this.spedytorID = spedytorID;
    }
}

```

7.2.10 Kierownik

```

package Model;

public class Kierownik extends Uzytkownik {

    private int spedytorID;
    private int magazynID;
    private int CzasZaladunku;

    public int getMagazynID() {
        return this.magazynID;
    }

    /**
     *
     * @param magazynID
     */
    public void setMagazynID(int magazynID) {
        this.magazynID = magazynID;
    }
}

```

```

    public int getCzasZaladunku() {
        return CzasZaladunku;
    }

    /**
     *
     * @param CzasZaladunku
     */
    public void setCzasZaladunku(int CzasZaladunku) {
        this.CzasZaladunku = CzasZaladunku;
    }

    public int getSpedytorID() {
        return this.spedytorID;
    }

    /**
     *
     * @param spedytorID
     */
    public void setSpedytorID(int spedytorID) {
        this.spedytorID = spedytorID;
    }

    public int getCzaasZaladunku() {
        return CzasZaladunku;
    }

    /**
     *
     * @param CzasZaladunku
     */
    public void setCzaasZaladunku(int CzasZaladunku) {
        this.CzasZaladunku = CzasZaladunku;
    }
}

```

7.2.11 Kierowca

```

package Model;

import java.util.*;

public class Kierowca extends Uzytkownik {
    private int spedytorID;

```

```

private ArrayList<Model.Zlecenie> ListaZlecen = new ArrayList<>();
private int CzasPracy;
private int DziennyLimit;
private ArrayList<Model.Powiadomienie> upomnienia = new ArrayList<>();
private int karaPieniezna;
private int CzasJazdyBezLimitu;

public int getCzasPracy() {
    return CzasPracy;
}

/**
 *
 * @param CzasPracy
 */
public void setCzasPracy(int CzasPracy) {
    this.CzasPracy = CzasPracy;
}

public int getDziennyLimit() {
    return DziennyLimit;
}

/**
 *
 * @param DziennyLimit
 */
public void setDziennyLimit(int DziennyLimit) {
    this.DziennyLimit = DziennyLimit;
}

public int getCzasJazdyBezLimitu() {
    return CzasJazdyBezLimitu;
}

/**
 *
 * @param CzasJazdyBezLimitu
 */
public void setCzasJazdyBezLimitu(int CzasJazdyBezLimitu) {
    this.CzasJazdyBezLimitu = CzasJazdyBezLimitu;
}

public int getSpedytorID() {
    return this.spedytorID;
}

/**

```

```

*
 * @param spedytorID
 */
public void setSpedytorID( int spedytorID ) {
    this.spedytorID = spedytorID ;
}

public ArrayList<Zlecenie> getListaZlecen() {
    return ListaZlecen ;
}

/***
 *
 * @param ListaZlecen
 */
public void setListaZlecen( ArrayList<Model.Zlecenie> ListaZlecen ) {
    this.ListaZlecen = ListaZlecen ;
}

public ArrayList<Model.Powiadomienie> getUpomnienia() {
    return this.upomnienia ;
}

/***
 *
 * @param upomnienia
 */
public void setUpomnienia( ArrayList<Powiadomienie> upomnienia ) {
    this.upomnienia = upomnienia ;
}

public int getKaraPieniezna() {
    return this.karaPieniezna ;
}

/***
 *
 * @param karaPieniezna
 */
public void setKaraPieniezna( int karaPieniezna ) {
    this.karaPieniezna = karaPieniezna ;
}

}

```

7.2.12 IDaneTymczasowe

```

package Model;

public class DaneTymczasowe implements IDaneTymczasowe {

    public Kierowca UtworzKierowce() {
        Kierowca kierowca = new Kierowca();
        return kierowca;
    }

    /**
     *
     * @param model
     */
    public Pojazd UtworzPojazd( String model) {
        Pojazd pojazd = new Pojazd(model);
        return pojazd;
    }

    public Spedytor UtworzSpedytora() {
        Spedytor spedytor = new Spedytor();
        return spedytor;
    }

    public Zlecenie UtworzZlecenie() {
        Zlecenie zlecenie = new Zlecenie();
        return zlecenie;
    }

    public Kierownik UtworzKierownika() {
        Kierownik kierownik = new Kierownik();
        return kierownik;
    }

    public Powiadomienie UtworzPowiadomienie() {
        Powiadomienie powiadomienie = new Powiadomienie();
        return powiadomienie;
    }
}

```

7.2.13 DaneTymczasowe

```

package Model;

public class DaneTymczasowe implements IDaneTymczasowe {

    public Kierowca UtworzKierowce() {

```

```
        return new Kierowca();
    }

    /**
     *
     * @param model
     */
    public Pojazd UtworzPojazd(String model) {
        return new Pojazd(model);
    }

    public Spedytor UtworzSpedytora() {
        return new Spedytor();
    }

    public Zlecenie UtworzZlecenie() {
        return new Zlecenie();
    }

    public Kierownik UtworzKierownika() {
        return new Kierownik();
    }

    public Powiadomienie UtworzPowiadomienie() {
        return new Powiadomienie();
    }

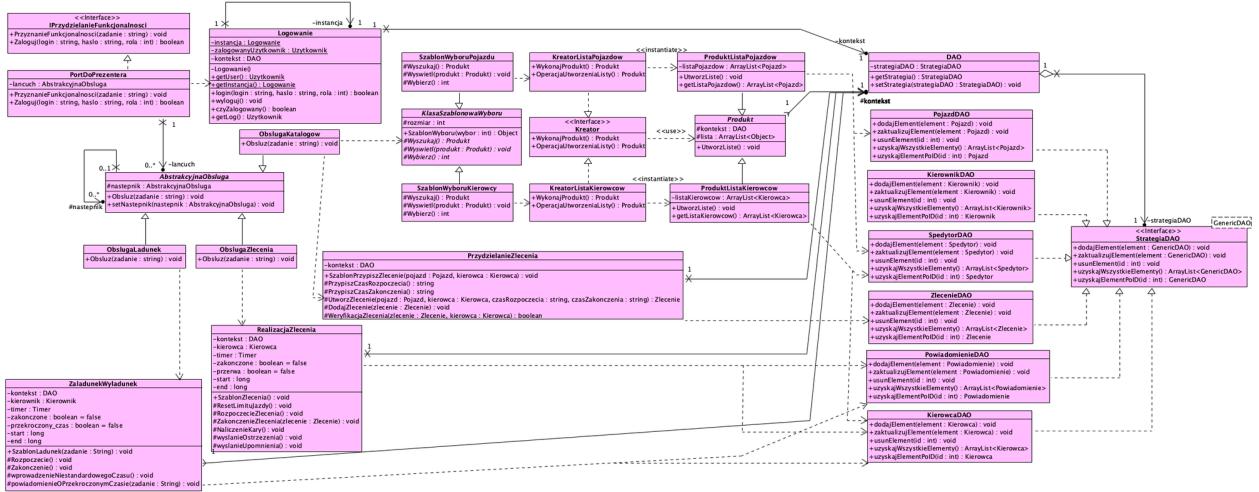
}
```

8 Warstwa Prezentera

Warstwa prezentera zarządza logiką aplikacji, przetwarza dane z modelu oraz udostępnia je widokowi. Jest odpowiedzialna za odbieranie danych od użytkownika i przekazywanie ich do modelu, jak również za zwracanie wyników z modelu do warstwy widoku.

8.0.1 Wzorce Projektowe

1. **Singleton (Logowanie)**: Zapewnia, że klasa Logowanie ma tylko jedną instancję w całej aplikacji i umożliwia globalny dostęp do niej.
2. **Fasada (PortDoPrezentera + IPrzydzielanieFunkcjonalności)**: Upraszczają dostępu do funkcjonalności prezentera, ukrywając złożoność implementacji wewnętrznej za jednym punktem dostępu.
3. **Łańcuch zobowiązań (AbstrakcyjnaObsługa + ObsługaKatalogu)**: Pozwala na przekazywanie żądań przez łańcuch obiektów w celu znalezienia tego, który obsłuży dane zadanie.
4. **Metoda szablonowa (SzablonWyboruPojazdu + KlasaSzablonowaWyboru + SzablonWyboruKierowcy)**: Definiuje szkielet algorytmu w klasie bazowej, a implementację szczegółów pozostawia klasom dziedziczącym.
5. **Metoda wytwarzająca (Interfejs Kreator wraz z klasami dziedziczącymi + Klasa abstrakcyjna Produkty + Klasy dziedziczące)**: Udostępnia interfejs do tworzenia obiektów w klasach potomnych, zamiast tworzyć instancje bezpośrednio.
6. **DAO (Klasa DAO)**: Oddziela warstwę logiki aplikacji od dostępu do danych trwałych, ułatwiając zarządzanie bazą danych i jej zmianami.
7. **Strategia (Interfejs StrategiaDAO + klasy go implementujące + klasa kontekstowa DAO)**: wzorzec projektowy, definiuje rodzinę algorytmów, w oddzielnych klasach i umożliwia ich dynamiczne wymienianie w trakcie działania programu. W połączeniu ze wzorcem DAO tworzy skalowalny łącznik pomiędzy prezenterem a danymi.



Rysunek 12: Diagram klas komponentu Prezenter

8.1 Kod Prezenter

8.2 PortDoPrezentera

```
package Prezenter;

public class PortDoPrezentera implements IPrzyzzielanieFunkcjonalnosci {

    private AbstrakcyjnaObsluga lancuch = new ObslugaKatalogow();

    /**
     *
     * @param zadanie
     */
    public void PrzyznanieFunkcjonalosci(String zadanie) {
        AbstrakcyjnaObsluga lancuch2 = new ObslugaZlecenia();
        AbstrakcyjnaObsluga lancuch3 = new ObslugaLadunek();

        lancuch.setNastepnik(lancuch2);
        lancuch2.setNastepnik(lancuch3);

        lancuch.Obsluz(zadanie);
    }

    /**
     *
     * @param login
     * @param haslo
     */
}
```

```

    * @param rola
    */
public boolean Zaloguj(String login, String haslo, int rola) {
    Logowanie sesja = Logowanie.getInstancja();
    boolean logged = sesja.login(login, haslo, rola);
    return logged;
}

}

```

8.2.1 AbstrakcyjnaObsluga

```

package Prezenter;

import java.util.*;

public abstract class AbstrakcyjnaObsluga {

    protected AbstrakcyjnaObsluga nastepnik;

    /**
     *
     * @param zadanie
     */
    public void Obsluz(String zadanie) {

    }

    /**
     *
     * @param nastepnik
     */
    public void setNastepnik(AbstrakcyjnaObsluga nastepnik) {
        this.nastepnik = nastepnik;
    }
}

```

8.2.2 DAO

```

package Prezenter;

public class DAO {

    private StrategiaDAO strategiaDAO;

    public StrategiaDAO getStrategia() {
        return this.strategiaDAO;
    }
}

```

```

        }

    /**
     *
     * @param strategiaDAO
     */
    public void setStrategia(StrategiaDAO strategiaDAO) {
        this.strategiaDAO = strategiaDAO;
    }

}

```

8.2.3 IPrzydzielanieFunkcjonalnosci

```

package Prezenter;

public interface IPrzydzielanieFunkcjonalnosci {

    /**
     *
     * @param zadanie
     */
    void PrzyznanieFunkcjonalnosci(String zadanie);

    /**
     *
     * @param login
     * @param haslo
     * @param rola
     */
    boolean Zaloguj(String login, String haslo, int rola);

}

```

8.2.4 KierowcaDAO

```

package Prezenter;

import Model.*;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class KierowcaDAO implements StrategiaDAO<Kierowca> {

```

```

    /**
     *
     * @param element
     */
    public void dodajElement(Kierowca element) {
        // TODO – implement Prezenter.KierowcaDAO.dodajElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param element
     */
    public void zaktualizujElement(Kierowca element) {
        // TODO – implement Prezenter.KierowcaDAO.zaktualizujElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param id
     */
    public void usunElement(int id) {
        // TODO – implement Prezenter.KierowcaDAO.usunElement
        throw new UnsupportedOperationException();
    }

    public ArrayList<Kierowca> uzyskajWszystkieElementy() {
        ArrayList<Kierowca> lista = new ArrayList<>();
        IDaneTymczasowe model = new DaneTymczasowe();
        String baza_dane = "/Applications/IOcode/Kierowcy.txt";
        try (BufferedReader reader = new BufferedReader
(new FileReader(baza_dane))) {
            String line;
            while((line = reader.readLine()) != null){
                Kierowca kierowca = model.UtworzKierowce();
                String[] wartosci = line.split(" ");
                kierowca.setUzytkownikID(Integer.parseInt(wartosci[0]));
                kierowca.setNazwaUzytkownika(wartosci[1]);
                kierowca.setHaslo(wartosci[2]);
                kierowca.setImie(wartosci[3]);
                kierowca.setNazwisko(wartosci[4]);
                kierowca.setSpedytorID(Integer.parseInt
(wartosci[5]));
                kierowca.setCzasPracy(Integer.parseInt
(wartosci[6]));
                kierowca.setDziennyLimit(Integer.parseInt
(wartosci[7]));
            }
        }
    }
}

```

```

        kierowca.setKaraPieniezna(Integer.parseInt
(wartosci[8]));
        lista.add(kierowca);
    }
} catch (IOException e) {
    e.printStackTrace();
}
return lista;
}

/**
 *
 * @param id
 */
public Kierowca uzyskajElementPoID(int id) {
    // TODO – implement Prezenter.KierowcaDAO.uzyskajElementPoID
    throw new UnsupportedOperationException();
}
}

```

8.2.5 KierownikDAO

```

package Prezenter;

import Model.*;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class KierownikDAO implements StrategiaDAO<Kierownik> {

    /**
     *
     * @param element
     */
    public void dodajElement(Kierownik element) {
        // TODO – implement Prezenter.KierownikDAO.dodajElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param element
     */

```

```

public void zaktualizujElement(Kierownik element) {
    // TODO – implement Prezenter.KierownikDAO.zaktualizujElement
    throw new UnsupportedOperationException();
}

/**
 *
 * @param id
 */
public void usunElement(int id) {
    // TODO – implement Prezenter.KierownikDAO.usunElement
    throw new UnsupportedOperationException();
}

public ArrayList<Kierownik> uzyskajWszystkieElementy() {
    ArrayList<Kierownik> lista = new ArrayList<>();
    IDaneTymczasowe model = new DaneTymczasowe();
    String baza_dane = "/Applications/IOcode/Kierownicy.txt";
    try (BufferedReader reader = new BufferedReader
        (new FileReader(baza_dane))) {
        String line;
        while((line = reader.readLine()) != null){
            Kierownik kierownik = model.UtworzKierownika();
            String [] wartosci = line.split(" ");
            kierownik.setUzytkownikID(Integer.parseInt
                (wartosci[0]));
            kierownik.setNazwaUzytkownika(wartosci[1]);
            kierownik.setHaslo(wartosci[2]);
            kierownik.setImie(wartosci[3]);
            kierownik.setNazwisko(wartosci[4]);
            kierownik.setSpedytorID(Integer.parseInt
                (wartosci[5]));
            kierownik.setMagazynID(Integer.parseInt
                (wartosci[6]));
            lista.add(kierownik);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return lista;
}

/**
 *
 * @param id
 */
public Kierownik uzyskajElementPoID(int id) {
    // TODO – implement Prezenter.KierownikDAO.uzyskajElementPoID
}

```

```

        throw new UnsupportedOperationException();
    }

}

```

8.2.6 PojazdDAO

```

package Prezenter;

import Model.DaneTymczasowe;
import Model.IDaneTymczasowe;
import Model.Pojazd;
import Model.PylekPojazd;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class PojazdDAO implements StrategiaDAO<Pojazd> {

    /**
     *
     * @param element
     */
    public void dodajElement(Pojazd element) {
        // TODO – implement Prezenter.PojazdDAO.dodajElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param element
     */
    public void zaktualizujElement(Pojazd element) {
        // TODO – implement Prezenter.PojazdDAO.zaktualizujElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param id
     */
    public void usunElement(int id) {
        // TODO – implement Prezenter.PojazdDAO.usunElement
        throw new UnsupportedOperationException();
    }
}

```

```

public ArrayList<Pojazd> uzyskajWszystkieElementy() {
    ArrayList<Pojazd> lista = new ArrayList<>();
    IDaneTymczasowe model = new DaneTymczasowe();
    String baza_dane = "/Applications/IOcode/Pojazdy.txt";
    try (BufferedReader reader = new BufferedReader
        (new FileReader(baza_dane))) {
        String line;
        while((line = reader.readLine()) != null){
            String [] wartosci = line.split(" ");
            Pojazd pojazd = model.UtworzPojazd(wartosci[3]);
            pojazd.setPojazdID(Integer.parseInt(wartosci[0]));
            pojazd.setSpedytorID(Integer.parseInt(wartosci[1]));
            pojazd.setStanPojazdu(Boolean.parseBoolean
                (wartosci[2]));
            pojazd.setPylek((PylekPojazd) pojazd.getFabryka()
                .getPylek(wartosci[3]));
            lista.add(pojazd);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return lista;
}

/**
 *
 * @param id
 */
public Pojazd uzyskajElementPoID(int id) {
    // TODO – implement Prezenter.PojazdDAO.uzyskajElementPoID
    throw new UnsupportedOperationException();
}
}

```

8.2.7 SpedytorDAO

```

package Prezenter;

import Model.DaneTymczasowe;
import Model.IDaneTymczasowe;
import Model.Spedytor;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

```

```

import java.util.ArrayList;

public class SpedytorDAO implements StrategiaDAO<Spedytor> {

    /**
     *
     * @param element
     */
    public void dodajElement(Spedytor element) {
        // TODO – implement Prezenter.SpedytorDAO.dodajElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param element
     */
    public void zaktualizujElement(Spedytor element) {
        // TODO – implement Prezenter.SpedytorDAO.zaktualizujElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param id
     */
    public void usunElement(int id) {
        // TODO – implement Prezenter.SpedytorDAO.usunElement
        throw new UnsupportedOperationException();
    }

    public ArrayList<Spedytor> uzyskajWszystkieElementy() {
        ArrayList<Spedytor> lista = new ArrayList<>();
        IDaneTymczasowe model = new DaneTymczasowe();
        String baza_dane = "/Applications/IOcode/Spedytorzy.txt";
        try (BufferedReader reader = new BufferedReader(
            new FileReader(baza_dane))) {
            String line;
            while((line = reader.readLine()) != null){
                Spedytor spedytor = model.UtworzSpedytora();
                String[] wartosci = line.split(" ");
                spedytor.setUzytkownikID(Integer.parseInt(wartosci[0]));
                spedytor.setNazwaUzytkownika(wartosci[1]);
                spedytor.setHaslo(wartosci[2]);
                spedytor.setImie(wartosci[3]);
                spedytor.setNazwisko(wartosci[4]);
                lista.add(spedytor);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return lista;
}

/**
 *
 * @param id
 */
public Spedytor uzyskajElementPoID(int id) {
    // TODO – implement Prezenter.SpedytorDAO.uzyskajElementPoID
    throw new UnsupportedOperationException();
}
}

```

8.2.8 StrategiaDAO

```

package Prezenter;

import java.util.ArrayList;

public interface StrategiaDAO<GenericDAO> {

    /**
     *
     * @param element
     */
    void dodajElement(GenericDAO element);

    /**
     *
     * @param element
     */
    void zaktualizujElement(GenericDAO element);

    /**
     *
     * @param id
     */
    void usunElement(int id);

    ArrayList<GenericDAO> uzyskajWszystkieElementy();
}

/**

```

```

    *
    * @param id
    */
GenericDAO uzyskajElementPoID( int id );

}

```

8.2.9 PowiadomienieDAO

```

package Prezenter;

import Model.DaneTymczasowe;
import Model.IDaneTymczasowe;
import Model.Powiadomienie;
import Model.RodzajPowiadomienia;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class PowiadomienieDAO implements StrategiaDAO<Powiadomienie> {

    /**
     *
     * @param element
     */
    public void dodajElement(Powiadomienie element) {
        String baza_dane = "/Applications/IOcode/Powiadomienia.txt";
        try (FileWriter writer = new FileWriter(baza_dane, true)) {
            writer.write(element.getPowiadomienieID() + " " +
                    element.getUzytkownikID() + " " +
                    element.getSpedytorID() + " " +
                    element.getTypPowiadomienia() + "\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     *
     * @param element
     */
    public void zaktualizujElement(Powiadomienie element) {
        // TODO – implement Prezenter.PowiadomienieDAO.zaktualizujElement
        throw new UnsupportedOperationException();
    }
}

```

```

}

/**
 *
 * @param id
 */
public void usunElement(int id) {
    // TODO – implement Prezenter.PowiadomienieDAO.usunElement
    throw new UnsupportedOperationException();
}

public ArrayList<Powiadomienie> uzyskajWszystkieElementy() {
    ArrayList<Powiadomienie> lista = new ArrayList<>();
    IDaneTymczasowe model = new DaneTymczasowe();
    String baza_dane = "/Applications/IOcode/Powiadomienia.txt";
    try (BufferedReader reader = new BufferedReader
        (new FileReader(baza_dane))) {
        String line;
        while((line = reader.readLine()) != null){
            Powiadomienie powiadomienie =
                model.UtworzPowiadomienie();
            String[] wartosci = line.split(" ");
            powiadomienie.setPowiadomienieID
                (Integer.parseInt(wartosci[0]));
            powiadomienie.setUzytkownikID
                (Integer.parseInt(wartosci[1]));
            powiadomienie.setSpedytorID
                (Integer.parseInt(wartosci[2]));
            powiadomienie.setTypPowiadomienia
                (RodzajPowiadomienia.valueOf(wartosci[3]));
            lista.add(powiadomienie);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return lista;
}

/**
 *
 * @param id
 */
public Powiadomienie uzyskajElementPoID(int id) {
    // TODO – implement Prezenter.PowiadomienieDAO.uzyskajElementPoID
    throw new UnsupportedOperationException();
}

```

8.2.10 ZlecenieDAO

```
package Prezenter;

import Model.DaneTymczasowe;
import Model.IDaneTymczasowe;
import Model.Zlecenie;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class ZlecenieDAO implements StrategiaDAO<Zlecenie> {

    /**
     *
     * @param element
     */
    public void dodajElement(Zlecenie element) {
        String baza_dane = "/Applications/IOcode/Zlecenia.txt";
        try (FileWriter writer = new FileWriter(baza_dane, true)) {
            writer.write(element.getZlecenieID() + " " +
                    element.getKierowcaID() + " " +
                    element.getPojazdID() + " " +
                    element.getCzasRozpoczecia() + " " +
                    element.getCzasZakonczenia() + "\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     *
     * @param element
     */
    public void zaktualizujElement(Zlecenie element) {
        // TODO – implement Prezenter.ZlecenieDAO.zaktualizujElement
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param id
     */
    public void usunElement(int id) {
        String baza_dane = "/Applications/IOcode/Zlecenia.txt";
```

```

        ArrayList<Zlecenie> lista = uzyskajWszystkieElementy();
        try (FileWriter writer = new FileWriter(baza_dane)) {
            for(Zlecenie element : lista){
                if(element.getZlecenieID() != id){
                    writer.write(element.getZlecenieID() +
                    " " + element.getKierowcaID() + " " +
                    element.getCzasRozpoczecia() + " " +
                    element.getCzasZakonczenia() + "\n");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public ArrayList<Zlecenie> uzyskajWszystkieElementy() {
        ArrayList<Zlecenie> lista = new ArrayList<>();
        IDaneTymczasowe model = new DaneTymczasowe();
        String baza_dane = "/Applications/IOcode/Zlecenia.txt";
        try (BufferedReader reader = new BufferedReader
        (new FileReader(baza_dane))) {
            String line;
            while((line = reader.readLine()) != null){
                Zlecenie zlecenie = model.UtworzZlecenie();
                String[] wartosci = line.split(" ");
                zlecenie.setZlecenieID(Integer.parseInt(wartosci[0]));
                zlecenie.setKierowcaID(Integer.parseInt(wartosci[1]));
                zlecenie.setPojazdID(Integer.parseInt(wartosci[2]));
                zlecenie.setCzasRozpoczecia(wartosci[3]);
                zlecenie.setCzasZakonczenia(wartosci[4]);
                lista.add(zlecenie);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return lista;
    }

    /**
     * @param id
     */
    public Zlecenie uzyskajElementPoID(int id) {
        // TODO - implement Prezenter.ZlecenieDAO.uzyskajElementPoID
        throw new UnsupportedOperationException();
    }
}

```

8.2.11 KlasaSzablonowaWyboru

```
package Prezenter;

import Model.Kierowca;
import Model.Pojazd;

public abstract class KlasaSzablonowaWyboru {

    protected int rozmiar = 0;

    /**
     *
     * @param wybor
     */
    public Object SzablonWyboru(int wybor) {
        Produkt produkt = Wyszukaj();
        Wyswietl(produkt);
        if(wybor == 2){
            int element_id = Wybierz();
            if(element_id == -1){
                return null;
            }
            if(produkt instanceof ProduktListaPojazdow){
                Pojazd pojazd = ((ProduktListaPojazdow)
                    produkt).getListaPojazdow().get(element_id);
                return pojazd;
            }
            else if(produkt instanceof ProduktListaKierowcow){
                Kierowca kierowca = ((ProduktListaKierowcow) produkt).
                    getListaKierowcow().get(element_id);
                return kierowca;
            }
        }
        return null;
    }

    protected abstract Produkt Wyszukaj();

    /**
     *
     * @param produkt
     */
    protected abstract void Wyswietl(Produkt produkt);

    protected abstract int Wybierz();
}
```

8.2.12 KreatorListaKierowcow

```
package Prezenter;

public class KreatorListaKierowcow implements Kreator {

    public Prezenter.Produkt WykonajProdukt() {
        ProduktListaKierowcow produkt = new ProduktListaKierowcow();
        return produkt;
    }

    public Prezenter.Produkt OperacjaUtworzeniaListy() {
        Produkt produkt = WykonajProdukt();
        produkt.UtworzListe();
        return produkt;
    }
}
```

8.2.13 KreatorListaPojazdow

```
package Prezenter;

public class KreatorListaPojazdow implements Kreator {

    public Prezenter.Produkt WykonajProdukt() {
        ProduktListaPojazdow produkt = new ProduktListaPojazdow();
        return produkt;
    }

    public Prezenter.Produkt OperacjaUtworzeniaListy() {
        Produkt produkt = WykonajProdukt();
        produkt.UtworzListe();
        return produkt;
    }
}
```

8.2.14 Logowanie

```
package Prezenter;

import Model.Uzytkownik;

import java.util.ArrayList;

public class Logowanie {
```

```

private static Logowanie instancja;
private static Uzytkownik zalogowanyUzytkownik;
private DAO kontekst = new DAO();

private Logowanie() {
}

public static Uzytkownik getUser() {
    return zalogowanyUzytkownik;
}

public static Logowanie getInstancja() {
    if (instancja == null) {
        instancja = new Logowanie();
    }
    return instancja;
}

/**
 *
 * @param login
 * @param haslo
 * @param rola
 */
public boolean login(String login, String haslo, int rola) {
    ArrayList<Uzytkownik> dane;

    switch(rola){
        case 1: KierowcaDAO kierowcaDAO = new KierowcaDAO();
                  kontekst.setStrategia(kierowcaDAO); break;
        case 2: KierownikDAO kierownikDAO = new KierownikDAO();
                  kontekst.setStrategia(kierownikDAO); break;
        case 3: SpedytorDAO spedytorDAO = new SpedytorDAO();
                  kontekst.setStrategia(spedytorDAO); break;
        default: break;
    }
    if(kontekst.getStrategia() != null) {
        dane = kontekst.getStrategia().uzyskajWszystkieElementy();
    }
    else{return false;}
    for(Uzytkownik element : dane){
        String userName = element.getNazwaUzytkownika();
        if(userName.equals(login)){
            String password = element.getHaslo();
            if(password.equals(haslo)){
                zalogowanyUzytkownik = element;
                return true;
            }
        }
    }
}

```

```

                else{ return false;}
            }
        }
    return false;
}

public void wyloguj() {
    // TODO – implement Prezenter.Logowanie.wyloguj
    throw new UnsupportedOperationException();
}

public boolean czyZalogowany() {
    // TODO – implement Prezenter.Logowanie.czyZalogowany
    throw new UnsupportedOperationException();
}

public Uzytkownik getLog() {
    // TODO – implement Prezenter.Logowanie.getLog
    throw new UnsupportedOperationException();
}
}

```

8.2.15 ObslugaKatalogow

```

package Prezenter;

import Model.Kierowca;
import Model.Pojazd;
import Widok.PortDoWidoku;

public class ObslugaKatalogow extends AbstrakcyjnaObsluga {

    /**
     * @param zadanie
     */
    public void Obsluz(String zadanie) {
        if(zadanie.equals("pojazdy")){
            // przegladanie pojazdow przez spedytora
            KlasaSzablonowaWyboru szablon =
            new SzablonWyboruPojazdu();
            new PortDoWidoku().wyswietlanie("Dostepne pojazdy:");
            szablon.SzablonWyboru(1);
        }
        else if(zadanie.equals("przydzial")){
            // przydzielenie zlecenia przez spedytora
        }
    }
}

```

```

(wlacznie z przegladaniem [ i wybraniem] pojazdu oraz kierowcy)
    KlasaSzablonowaWyboru szablon1 =
new SzablonWyboruKierowcy();
    new PortDoWidoku().wyswietlanie
("Dostepni kierowcy :");
    Kierowca kierowca = (Kierowca)
szablon1.SzablonWyboru(2);
    if(kierowca == null){
        return;
    }

Pojazd pojazd;
boolean sprawny;
do {
    KlasaSzablonowaWyboru szablon =
new SzablonWyboruPojazdu();
    new PortDoWidoku().wyswietlanie("Dostepne pojazdy -
wybierz pojazd o dobrym stanie technicznym :");
    pojazd = (Pojazd)szablon .SzablonWyboru(2);
    if(pojazd == null){
        return;
    }
    sprawny = pojazd.getStanPojazdu();
} while (!sprawny);

PrzydzielanieZlecenia przydzielanieZlecenia =
new PrzydzielanieZlecenia();
    przydzielanieZlecenia .SzablonPrzypiszZlecenie
(pojazd , kierowca);
}
else if(nastepnik != null){
    nastepnik .Obsluz(zadanie);
}
}
}
}

```

8.2.16 Obslugaladunek

```

package Prezenter;

public class ObslugaLadunek extends AbstrakcyjnaObsluga {

/**
*
* @param zadanie
*/

```

```

        public void Obsluz(String zadanie) {
            if(zadanie.equals("zaladunek") || zadanie.equals("wyladunek")){
                // przeprowadzenie zaladunku/wyladunku przez kierownika
                ZaladunekWyladunek ladunek = new ZaladunekWyladunek();
                ladunek.SzablonLadunek(zadanie);
            }
        }
    }
}

```

8.2.17 ObslugaZlecenia

```

package Prezenter;

public class ObslugaZlecenia extends AbstrakcyjnaObsluga {

    /**
     *
     * @param zadanie
     */
    public void Obsluz(String zadanie) {
        if(zadanie.equals("zlecenie")){
            // wykonanie zlecenia przez kierowce
            RealizacjaZlecenia realizacjaZlecenia =
            new RealizacjaZlecenia();
            realizacjaZlecenia.SzablonZlecenia();
        }
        else if(nastepnik != null){
            nastepnik.Obsluz(zadanie);
        }
    }
}

```

8.2.18 Produkt

```

package Prezenter;

import java.util.ArrayList;

public abstract class Produkt {

    protected DAO kontekst = new DAO();
    protected ArrayList<Object> lista = new ArrayList<>();

    public void UtworzListe() {
        // TODO – implement Prezenter.Produkt.UtworzListe
    }
}

```

```

        throw new UnsupportedOperationException();
    }
}

```

8.2.19 ProduktListaKierowcow

```

package Prezenter;

import Model.Kierowca;
import Model.Spedytor;

import java.util.ArrayList;

public class ProduktListaKierowcow extends Produkt {

    private ArrayList<Kierowca> listaKierowcow;

    public void UtworzListe() {
        Spedytor spedytor = (Spedytor) Logowanie.getUser();
        KierowcaDAO kierowcaDAO = new KierowcaDAO();
        kontekst.setStrategia(kierowcaDAO);

        listaKierowcow = kontekst.getStrategia().
uzyskajWszystkieElementy();
        for (Kierowca element : listaKierowcow){
            if (element.getSpedytorID() == spedytor.getUzytkownikID()) {
                lista.add(element);
            }
        }
    }

    public ArrayList<Kierowca> getListaKierowcow() {
        return this.listaKierowcow;
    }

}

```

8.2.20 ProduktListaPojazdow

```

package Prezenter;

import Model.Pojazd;
import Model.Spedytor;

import java.util.ArrayList;

public class ProduktListaPojazdow extends Produkt {

```

```

private ArrayList<Pojazd> listaPojazdow;

public void UtworzListe() {
    Spedytor spedytor = (Spedytor) Logowanie.getUser();
    PojazdDAO pojazdDAO = new PojazdDAO();
    kontekst.setStrategia(pojazdDAO);

    listaPojazdow = kontekst.getStrategia().
uzyskajWszystkieElementy();
    for(Pojazd element : listaPojazdow){
        if(element.getSpedytorID() == spedytor.getUzytkownikID()) {
            lista.add(element);
        }
    }
}

public ArrayList<Model.Pojazd> getListaPojazdow() {
    return this.listaPojazdow;
}

}

```

8.2.21 PrzydzielanieZlecenia

```

package Prezenter;

import Model.*;
import Widok.PortDoWidoku;

import java.util.ArrayList;

public class PrzydzielanieZlecenia {

    private DAO kontekst = new DAO();

    /**
     *
     * @param pojazd
     * @param kierowca
     */
    public void SzablonPrzypiszZlecenie(Pojazd pojazd ,
    Kierowca kierowca) {
        String rozpoczęcie = PrzypiszCzasRozpoczecia();
        String zakończenie = PrzypiszCzasZakonczenia();
        Zlecenie zlecenie = UtworzZlecenie(pojazd , kierowca ,
        rozpoczęcie , zakończenie);
    }
}

```

```

        boolean zweryfikowane = WeryfikacjaZlecenia(zlecenie , kierowca);
        if (zweryfikowane){
            new PortDoWidoku().wyswietlanie("Zlecenie z o s t a
o dodane dla kierowcy");
            DodajZlecenie(zlecenie);
        }
        else{
            new PortDoWidoku().wyswietlanie("Kierowca nie
moze zrealizowac tego zlecenia w swoim czasie pracy");
        }
    }

protected String PrzypiszCzasRozpoczecia() {
    int godzina , minuta;
    do{
        godzina = Integer.parseInt(new PortDoWidoku().
wprowadzanieDanych("Podaj godzine rozpoczecia: "));
        }while(godzina < 0 || godzina > 23);
    do{
        minuta = Integer.parseInt(new PortDoWidoku().
wprowadzanieDanych("Podaj minute rozpoczecia: "));
        }while(minuta < 0 || minuta > 59);

    return godzina + ":" + minuta;
}

protected String PrzypiszCzasZakonczenia() {
    int godzina , minuta;
    do{
        godzina = Integer.parseInt(new PortDoWidoku().
wprowadzanieDanych("Podaj godzine zakonczenia: "));
        }while(godzina < 0 || godzina > 23);
    do{
        minuta = Integer.parseInt(new PortDoWidoku().
wprowadzanieDanych("Podaj minute zakonczenia: "));
        }while(minuta < 0 || minuta > 59);

    return godzina + ":" + minuta;
}

/**
 *
 * @param pojazd
 * @param kierowca
 * @param czasRozpoczecia
 * @param czasZakonczenia
 */

```

```

    protected Zlecenie UtworzZlecenie(Pojazd pojazd ,
Kierowca kierowca , String czasRozpoczecia , String czasZakonczenia) {
        IDaneTymczasowe model = new DaneTymczasowe();
        Zlecenie zlecenie = model.UtworzZlecenie();
        int user_id = kierowca.getUzytkownikID();
        int pojazd_id = pojazd.getPojazdID();
        zlecenie.setKierowcaID(user_id);
        zlecenie.setPojazdID(pojazd_id);
        zlecenie.setCzasRozpoczecia(czasRozpoczecia);
        zlecenie.setCzasZakonczenia(czasZakonczenia);
        return zlecenie;
    }

    /**
     *
     * @param zlecenie
     */
    protected void DodajZlecenie(Zlecenie zlecenie) {
        int zlecenie_id;
        ZlecenieDAO zlecenieDAO = new ZlecenieDAO();
        kontekst.setStrategia(zlecenieDAO);
        StrategiaDAO strategia = kontekst.getStrategia();
        ArrayList<Zlecenie> zlecenia = strategia.
        uzyskajWszystkieElementy();
        if(zlecenia.isEmpty()) {zlecenie_id = 0;}
        else {zlecenie_id = zlecenia.getLast().getZlecenieID() + 1;}
        zlecenie.setZlecenieID(zlecenie_id);
        strategia.dodajElement(zlecenie);
    }

    /**
     *
     * @param zlecenie
     * @param kierowca
     */
    protected boolean WeryfikacjaZlecenia(Zlecenie
zlecenie , Kierowca kierowca) {
        int czasPracy = kierowca.getCzasPracy();
        String [] czasStart = zlecenie.getCzasRozpoczecia().
split(":");
        String [] czasKoniec = zlecenie.getCzasZakonczenia().
split(":");
        int godziny = Integer.parseInt(czasKoniec[0]) -
Integer.parseInt(czasStart[0]);
        int minuty = Integer.parseInt(czasKoniec[1]) -
Integer.parseInt(czasStart[1]);
        // czy zlecenie przekracza limit pracy (9h)
wyrazony w minutach
    }
}

```

```

        boolean czy_przekracza_limit = 540 >=
        ((godziny + 24) % 24) + ((minuty + 60) % 60 + czasPracy);
        return czy_przekracza_limit;
    }

}

```

8.2.22 RealizacjaZlecenia

```

package Prezenter;

import Model.*;
import Widok.PortDoWidoku;

import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;

public class RealizacjaZlecenia {

    private DAO kontekst = new DAO();
    private Kierowca kierowca = (Kierowca) Logowanie.getUser();
    private Timer timer;
    private boolean zakoñzone = false;
    private boolean przerwa = false;
    private long start;
    private long end;

    public void SzablonZlecenia() {
        PowiadomienieDAO powiadomienieDAO = new PowiadomienieDAO();
        kontekst.setStrategia(powiadomienieDAO);
        ArrayList<Powiadomienie> lista1 = kontekst.getStrategia().
uzyskajWszystkieElementy();
        ArrayList<Powiadomienie> upomnienia = new ArrayList<>();
        for (Powiadomienie element : lista1){
            if(element.getUzytkownikID() == kierowca.
getUzytkownikID()){
                upomnienia.add(element);
            }
        }
        kierowca.setUpomnienia(upomnienia);

        ZlecenieDAO zlecenieDAO = new ZlecenieDAO();
        kontekst.setStrategia(zlecenieDAO);
        ArrayList<Zlecenie> lista2 = kontekst.getStrategia().
uzyskajWszystkieElementy();
    }
}

```

```

ArrayList<Zlecenie> zlecenia = new ArrayList<>();
for (Zlecenie element : lista2) {
    if (element.getKierowcaID() ==
        kierowca.getUzytkownikID())){
        zlecenia.add(element);
    }
}
kierowca.setListaZlecen(zlecenia);

if (!zlecenia.isEmpty()) {
    Zlecenie zlecenie = zlecenia.getFirst();
    String action;
    do {
        ResetLimituJazdy();
        RozpoczecieZlecenia();
        action = new PortDoWidoku().wprowadzanieDanych
        ("(1) Udanie sie na przerwe\n(0) Zakoncz zlecenie\n");
        if (action.equals("1")){
            przerwa = true;
            new PortDoWidoku().wprowadzanieDanych
            ("<Enter> Wznowienie jazdy:");
            przerwa = false;
        }
    } while (!action.equals("0"));
    ZakonczenieZlecenia(zlecenie);
    return;
}
new PortDoWidoku().wyswietlanie
("Brak zlecan do wykonania");
}

protected void ResetLimituJazdy() {
    kierowca.setCzasJazdyBezLimitu(0);
}

protected void RozpoczecieZlecenia() {
    start = System.nanoTime();
    timer = new Timer();

    timer.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
            if (przerwa){
                timer.cancel();
            }
            else if (!zakonczone) {
                kierowca.setCzasJazdyBezLimitu(kierowca.
                    getCzasJazdyBezLimitu())+15);
            }
        }
    });
}

```

```

        System.out.println(
kierowca .getCzasJazdyBezLimitu()
/60 + " godzin i "+
kierowca .getCzasJazdyBezLimitu()
%60 + " minut ciaglej jazdy!";
);
}
},
}, 1500, 1500);

timer .schedule (new TimerTask() {
    @Override
    public void run() {
        if(przerwa){
            timer.cancel();
        }
        else if (!zakonczzone) {
            wyslanieOstrzezenia();
        }
    }
},
}, 24000);

timer .schedule (new TimerTask() {
    @Override
    public void run() {
        if(przerwa){
            timer.cancel();
        }
        else if (!zakonczzone) {
            wyslanieUpomnienia();
            if(kierowca .getUpomnienia().size () >= 5){
                NaliczenieKary();
            }
        }
    }
},
}, 27000);
}

/**
 *
 * @param zlecenie
 */
protected void ZakonczenieZlecenia(Zlecenie zlecenie) {
    end = System.nanoTime();
    zakonczzone = true;
    //if (timer != null) {
        timer.cancel();
        new PortDoWidoku().wyswietlanie("Ukonczenie zlecenia!");
    }
}

```

```

        int czasPracy = kierowca.getCzasPracy() + (int)
(end - start)/1000000000;
        kierowca.setCzasPracy(czasPracy);
        ArrayList<Zlecenie> nowaLista = kierowca.getListaZlecen();
        nowaLista.remove(zlecenie);
        kierowca.setListaZlecen(nowaLista);
        ZlecenieDAO zlecenieDAO = new ZlecenieDAO();
        kontekst.setStrategia(zlecenieDAO);
        int id = zlecenie.getZlecenieID();
        kontekst.getStrategia().usunElement(id);
    //}
}

protected void NaliczenieKary() {
    new PortDoWidoku().wyswietlanie("Naliczona zostala kara 100zl!");
    kierowca.setKaraPieniezna(kierowca.getKaraPieniezna() + 100);
}

protected void wyslanieOstrzezenia() {
    new PortDoWidoku().wyswietlanie("Przekroczone zostaly
4h jazdy. Rozwaz przerwe!");
}

protected void wyslanieUpomnienia() {
    int powiadomienie_id;
    new PortDoWidoku().wyswietlanie("Przekroczyony zostal
limit 4.5h! Zjedz na przerwe!");
    PowiadomienieDAO powiadomienieDAO = new PowiadomienieDAO();
    IDaneTymczasowe model = new DaneTymczasowe();
    kontekst.setStrategia(powiadomienieDAO);
    Powiadomienie powiadomienie = model.UtworzPowiadomienie();
    ArrayList<Powiadomienie> powiadomienia =
kontekst.getStrategia().uzyskajWszystkieElementy();
    if (powiadomienia.isEmpty()) { powiadomienie_id = 0; }
    else { powiadomienie_id = powiadomienia.getLast().
getPowiadomienieID() + 1; }
    powiadomienie.setPowiadomienieID(powiadomienie_id);

    int id_uzytkownika = kierowca.getUzytkownikID();
    int id_spedytora = kierowca.getSpedytorID();
    powiadomienie.setUzytkownikID(id_uzytkownika);
    powiadomienie.setSpedytorID(id_spedytora);
    powiadomienie.setTypPowiadomienia
(RodzajPowiadomienia.przekroczyonyLimit);
    kontekst.getStrategia().dodajElement(powiadomienie);
    ArrayList<Powiadomienie> upomnienia =
kierowca.getUpomnienia();
    upomnienia.add(powiadomienie);
}

```

```

        kierowca.setUpomnienia(upomnienia);
    }
}
```

8.2.23 SzablonWyboruKierowcy

```

package Prezenter;

import Model.Kierowca;
import Widok.PortDoWidoku;

import java.util.ArrayList;

public class SzablonWyboruKierowcy extends KlasaSzablonowaWyboru {

    protected Produkt Wyszukaj() {
        Kreator kreator = new KreatorListaKierowcow();
        return kreator.OperacjaUtworzeniaListy();
    }

    /**
     *
     * @param produkt
     */
    protected void Wyswietl(Produkt produkt) {
        int i = 1;
        ArrayList<Kierowca> kierowcy = ((ProduktListaKierowcow)
produkt).getListaKierowcow();
        for(Kierowca element : kierowcy){
            new PortDoWidoku().wyswietlanie(i + " " +
element.getImie() + " " + element.getNazwisko());
            i++;
        }
        rozmiar = kierowcy.size();
    }

    protected int Wybierz() {
        int index;
        do{
            index = Integer.parseInt(new PortDoWidoku().
wprowadzanieDanych("Wybierz numer kierowcy lub 0 aby wyjsc:"));
        }while(index < 0 || index > rozmiar);
        return —index;
    }

}
```

8.2.24 SzablonWyboruPojazdu

```
package Prezenter;

import Model.Pojazd;
import Model.PylekPojazd;
import Widok.PortDoWidoku;

import java.util.ArrayList;

public class SzablonWyboruPojazdu extends KlasaSzablonowaWyboru {

    protected Produkt Wyszukaj() {
        Kreator kreator = new KreatorListaPojazdow();
        return kreator.OperacjaUtworzeniaListy();
    }

    /**
     *
     * @param produkt
     */
    protected void Wyswietl(Prezenter.Punkt produkt) {
        ArrayList<Pojazd> pojazdy = ((ProduktListaPojazdow)
produkt).getListaPojazdow();
        int i = 1;
        for(Pojazd element : pojazdy){
            String stan;
            boolean sprawny = element.getStanPojazdu();
            if(sprawny){stan = "dobry";}
            else{stan = "wymaga naprawy";}
            PylekPojazd pylek = element.getPylek();
            new PortDoWidoku().wyswietlanie(i + " "
+ pylek.getModel() + " " + stan);
            i++;
        }
        rozmiar = pojazdy.size();
    }

    protected int Wybierz() {
        int index;
        do{
            index = Integer.parseInt(new PortDoWidoku().
wprowadzanieDanych("Wybierz numer pojazdu lub 0 aby wyjsc:"));
        }while(index < 0 || index > rozmiar);
        return —index;
    }
}
```

8.2.25 ZaladunekWyladunek

```
package Prezenter;

import Model.*;
import Widok.PortDoWidoku;

import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;

public class ZaladunekWyladunek {

    private DAO kontekst = new DAO();
    private Kierownik kierownik =
(Kierownik)Logowanie.getUser();
    private Timer timer;
    private boolean zakonczone = false;
    private boolean przekroczy_czas = false;
    private long start;
    private long end;

    public void SzablonLadunek(String zadanie) {
        Rozpoczenie();
        new PortDoWidoku().wprowadzanieDanych
("<Enter> Zakoncz " + zadanie + ": ");
        Zakonczenie();
        if(przekroczy_czas){
            wprowadzenieNiestandardowegoCzasu();
            powiadomienieOPrzekroczeniuCzasie(zadanie);
        }
    }

    protected void Rozpoczenie() {
        start = System.nanoTime();
        timer = new Timer();

        timer.schedule(new TimerTask(){
            @Override
            public void run() {
                if (!zakonczone) {
                    new PortDoWidoku().wyswietlanie
("Przekroczy zostal standardowy czas 1h");
                    przekroczy_czas = true;
                }
            }
        }, 5000); // 5 sekund
    }
}
```

```

protected void Zakonczenie() {
    end = System.nanoTime();
    zakonczone = true;
    if (timer != null) {
        timer.cancel();
        new PortDoWidoku().wyswietlanie("Ukonczenie zlecenia!");
    }
}

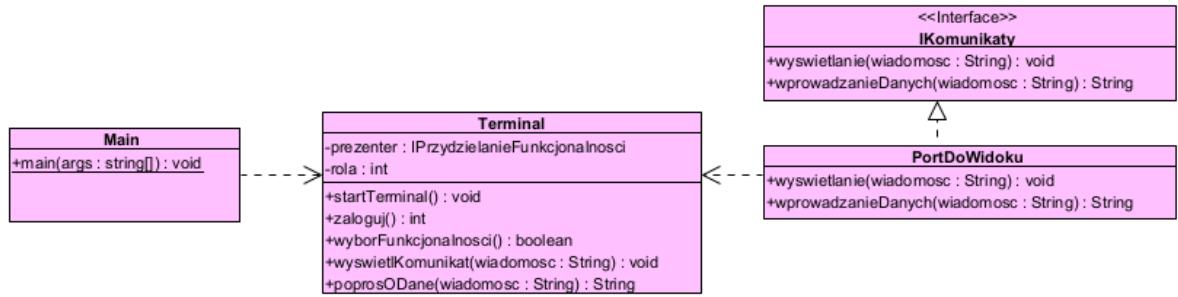
protected void wprowadzenieNiestandardowegoCzasu() {
    new PortDoWidoku().wyswietlanie("Opoznenie wynioslo " +
((end-start)/1000000000 - 5) + " minut i
zostanie\nuwzglednione w calkowitym czasie pracy kierowcy!");
}

protected void powiadomienieOPrzekroczeniemCzasie(String zadanie) {
    PowiadomienieDAO powiadomienieDAO = new PowiadomienieDAO();
    IDaneTymczasowe model = new DaneTymczasowe();
    Powiadomienie powiadomienie = model.UtworzPowiadomienie();
    kontekst.setStrategia(powiadomienieDAO);
    if(zadanie.equals("zaladunek")){
        powiadomienie.setTypPowiadomienia(RodzajPowiadomienia.
przekroczonyCzasZaladunku);
    }
    else if(zadanie.equals("wyladunek")){
        powiadomienie.setTypPowiadomienia(RodzajPowiadomienia.
przekroczonyCzasWyladunku);
    }
    StrategiaDAO strategia = kontekst.getStrategia();
    ArrayList<Powiadomienie> powiadomienia = strategia.
uzyskajWszystkieElementy();
    powiadomienie.setPowiadomienieID(powiadomienia.size());
    int id_kierownika = kierownik.getUzytkownikID();
    powiadomienie.setUzytkownikID(id_kierownika);
    int id_spedytora = kierownik.getSpedytorID();
    powiadomienie.setSpedytorID(id_spedytora);
    strategia.dodajElement(powiadomienie);
}
}

```

9 Warstwa Widoku

Warstwa widoku odpowiada za prezentację danych użytkownikowi oraz odbieranie jego interakcji. Widok komunikuje się wyłącznie z warstwą prezentera, dostarczając dane wejściowe i aktualizując wyświetlane informacje.



Rysunek 13: Diagram klas komponentu Widok

9.1 Kod Widok

9.1.1 Main

```
package Widok;

public class Main {

    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        Terminal terminal = new Terminal();
        terminal.startTerminal();
    }
}
```

9.1.2 IKomunikaty

```
package Widok;

public interface IKomunikaty {

    /**
     *
     */
}
```

```

    * @param wiadomosc
    */
void wyswietlanie(String wiadomosc);

String wprowadzanieDanych(String wiadomosc);

}

```

9.1.3 PortDoWidoku

```

package Widok;

public class PortDoWidoku implements IKomunikaty {

    /**
     *
     * @param wiadomosc
     */
    public void wyswietlanie(String wiadomosc) {
        new Terminal().wyswietlKomunikat(wiadomosc);
    }

    public String wprowadzanieDanych(String wiadomosc) {
        return new Terminal().poprosODane(wiadomosc);
    }

}

```

9.1.4 Terminal

```

package Widok;

import Prezenter.*;
import java.util.Scanner;

public class Terminal {

    private IPrzydzielanieFunkcjalnosci prezenter
    = new PortDoPrezentera();
    private int rola;

    public void startTerminal() {
        // proces logowania
        int wybor;
        do{
            wybor = zaloguj();

```

```

        if(wybor == 0) {return;}
    }while(wybor == 1);
System.out.println("Zalogowano!");
// korzystanie z systemu
boolean usingSystem;
do{
    usingSystem = wyborFunkcjalnosci();
}while(usingSystem);
}

public int zaloguj() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("<any key>) Wprowadzenie danych
logowania\n(0) Wyjscie\n");
    String wybor = scanner.nextLine();
    if(wybor.equals("0")) {return 0;}
    do{
        System.out.println("Rola uzytkownika:\n(1)
Kierowca\n(2)
Kierownik magazynu\n(3) Spedytor\n");
        rola = scanner.nextInt();
    }while(rola < 1 || rola > 3);
    scanner.nextLine();
    System.out.println("Nazwa uzytkownika: ");
    String username = scanner.nextLine();
    System.out.println("Haslo: ");
    String password = scanner.nextLine();

    boolean logged = prezenter.Zaloguj
(username, password, rola);
    if(logged) {return 2;}
    System.out.println("Wprowadzone dane sa niewlasciwe!");
    return 1;
}

public boolean wyborFunkcjalnosci() {
    Scanner scanner = new Scanner(System.in);
    String funkcjalnosc, zadanie = "";
    switch (rola){
        case 1:
            System.out.println("(0) Wyjscie\n(1)
Wykonanie zlecenia\n");
            funkcjalnosc = scanner.nextLine();
            if(funkcjalnosc.equals("1")){zadanie = "zlecenie";}
            break;
        case 2:
            System.out.println("(0) Wyjscie\n(1)
Przeprowadzenie zaladunku\n");
    }
}

```

```

        Przeprowadzenie wyladunku\n");
        funkcjonalosc = scanner.nextLine();
        if(funkcjonalosc.equals("1")){zadanie = "zaladunek";}
        else if(funkcjonalosc.equals("2"))
        {zadanie = "wyladunek";}
        break;
    case 3:
        System.out.println("(0) Wyjscie\n(1)
Przegladanie pojazdow\n(2)
Przydzielanie zlecen\n");
        funkcjonalosc = scanner.nextLine();
        if(funkcjonalosc.equals("1")){zadanie = "pojazdy";}
        else if(funkcjonalosc.equals("2"))
        {zadanie = "przydzial";}
        break;
    default: return false;
}
if(!funkcjonalosc.equals("0")){
    if(!zadanie.isEmpty()){
        prezenter.PrzyznanieFunkcjalnosci(zadanie);
    }
    return true;
}
return false;
}

/**
 *
 * @param wiadomosc
 */
public void wyswietlKomunikat(String wiadomosc) {
    System.out.println(wiadomosc);
}

public String poprosODane(String wiadomosc) {
    Scanner scanner = new Scanner(System.in);
    System.out.println(wiadomosc);
    return scanner.nextLine();
}
}

```

10 Przykładowe dane

Poniżej znajdują się przykładowe dane, które używane są w celu symulowania korzystania z bazy danych.

10.1 Dane Kierowców

```
0 kierowca1 12345 Jan Kowalski 3 0 540 0
1 kierowca2 qwerty Damian Nowak 3 0 540 0
```

10.2 Dane Kierowników magazynu

```
2 kierownik1 wasd Kamil Wicher 3 0 60
```

10.3 Dane Spedytorzy

```
3 spedytor1 zxvc Patryk Gawlas
```

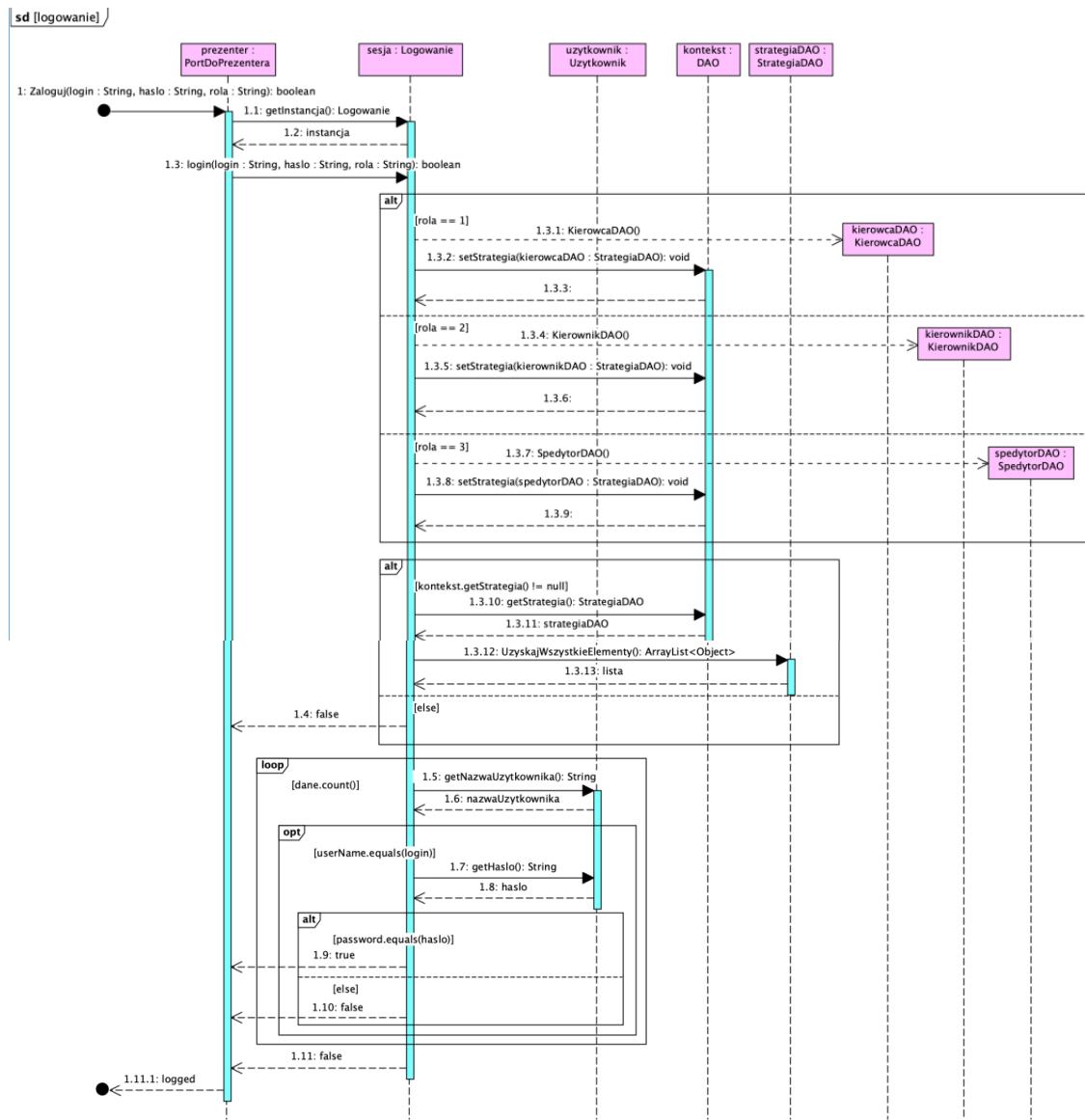
10.4 Dane Pojazdów

```
0 3 true Peugeot—Partner—Caddy—Berlingo
```

10.5 Dane powiadomienia

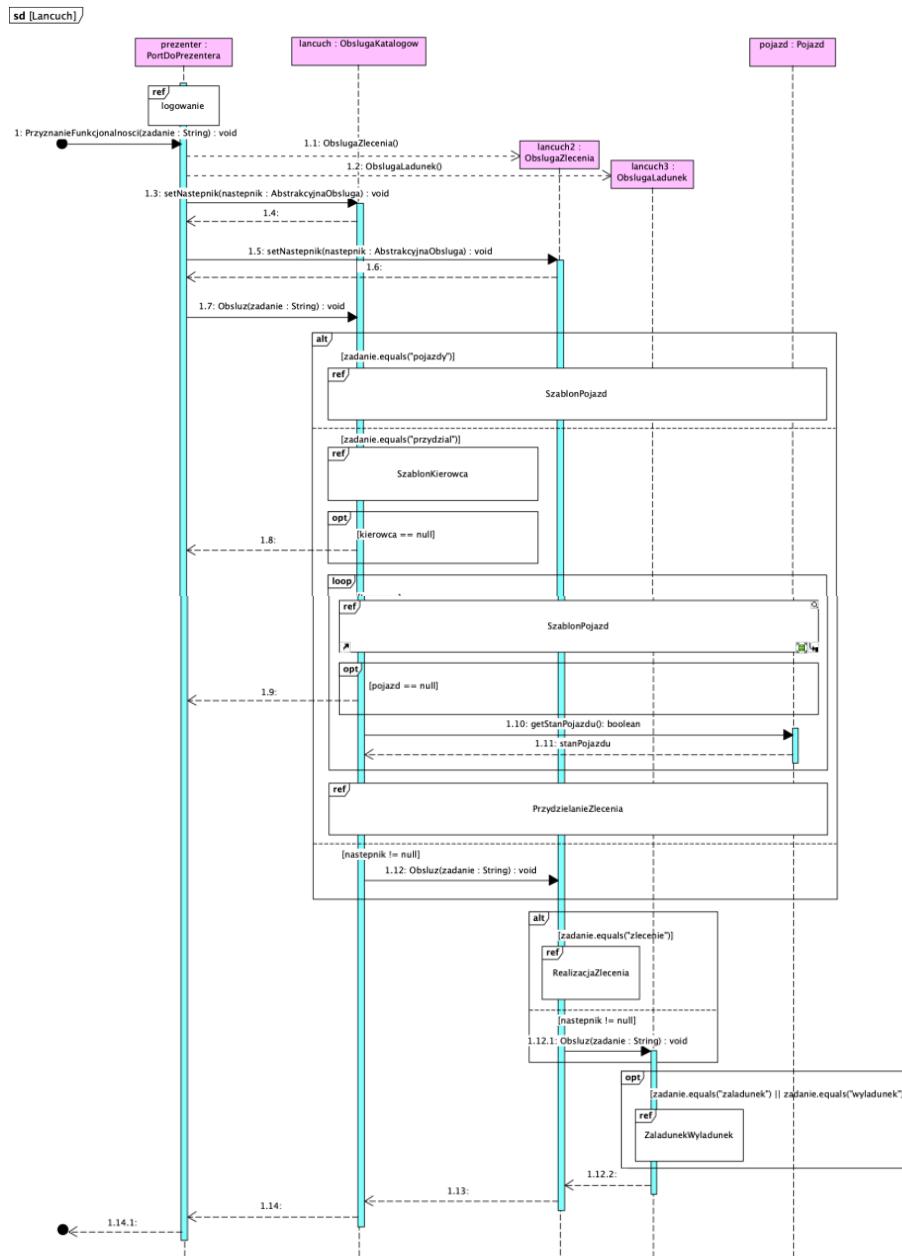
```
0 0 przekroczonyLimit
1 2 przekroczonyCzasZaladunku
```

11 Diagram Sekwencji dla logowania

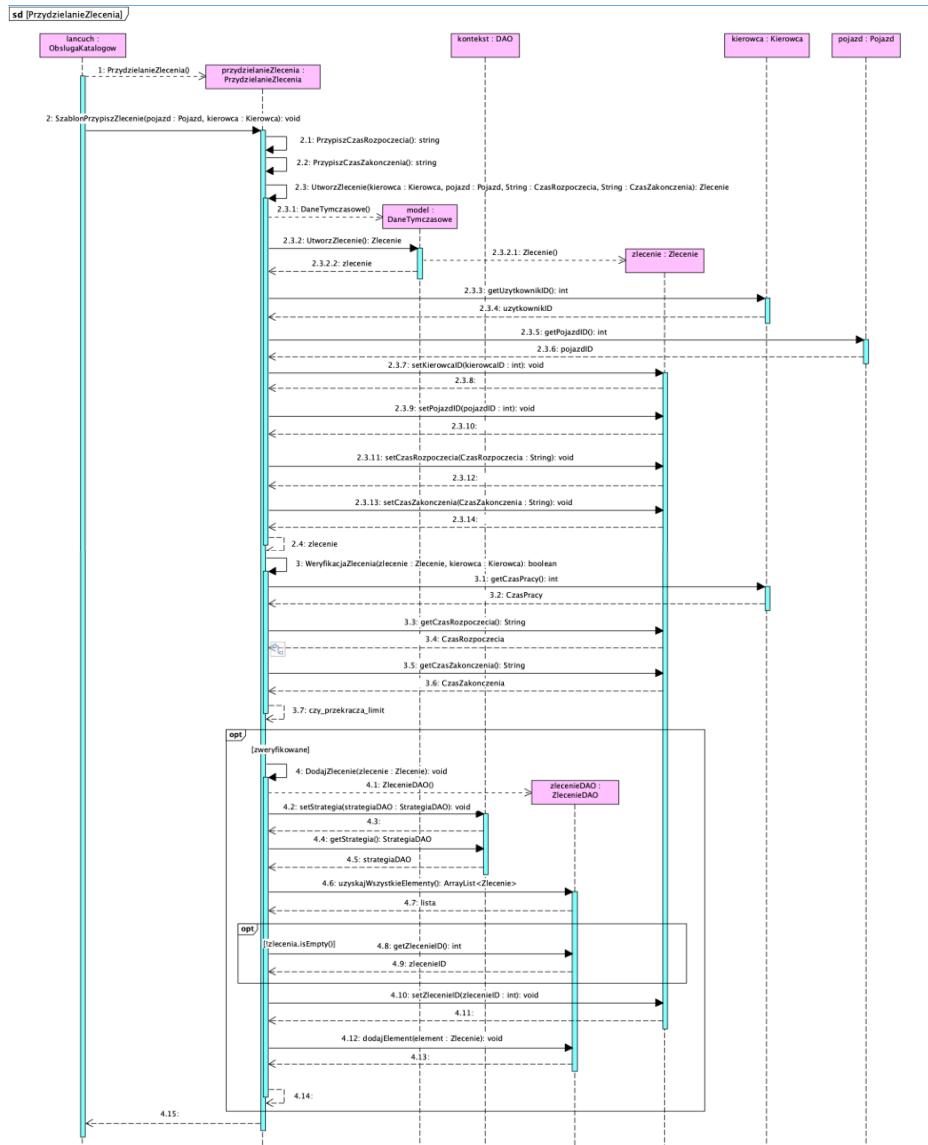


Rysunek 14: Diagram sekwencji dla czynności logowania

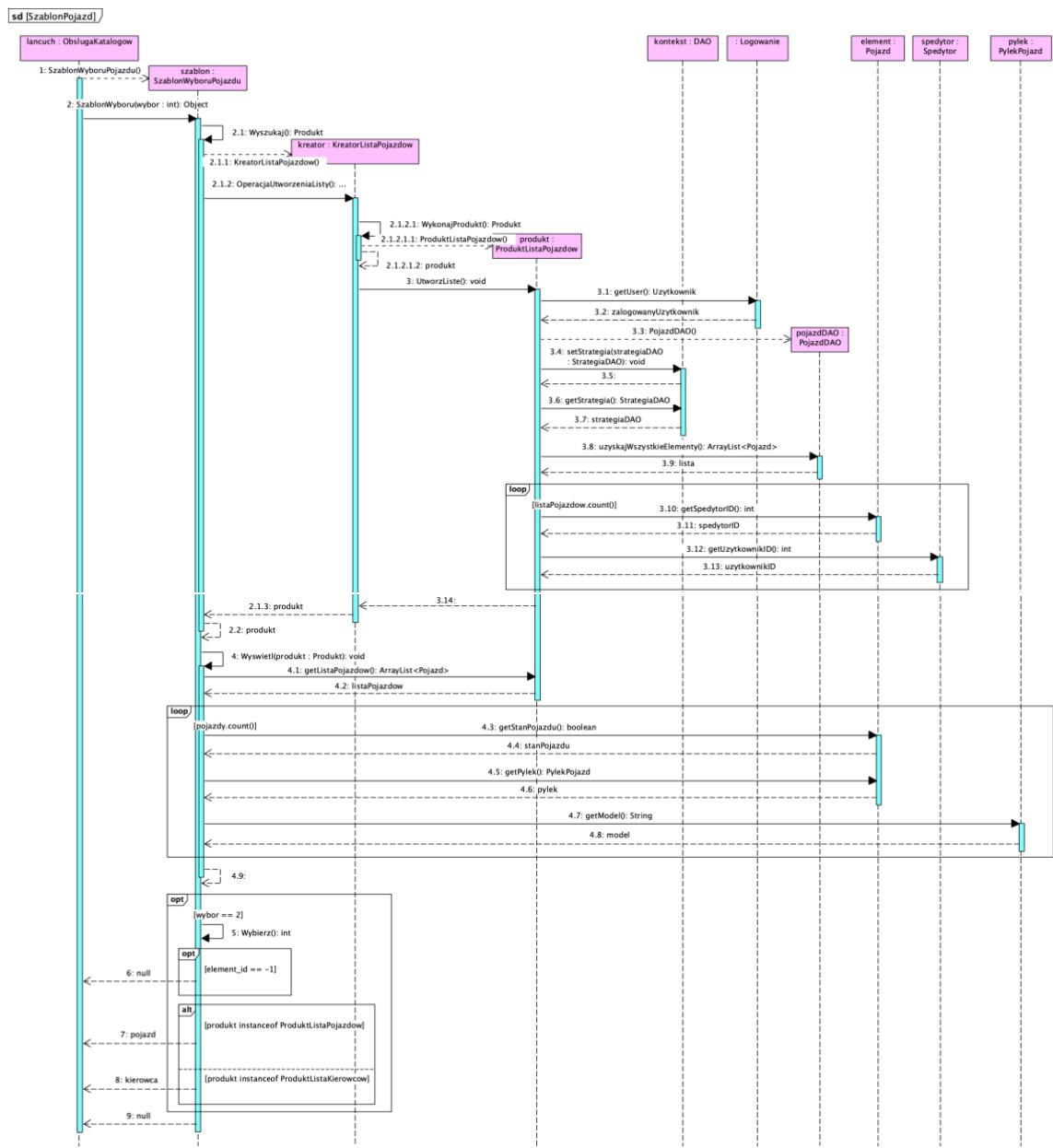
12 Złożony diagram sekwencji



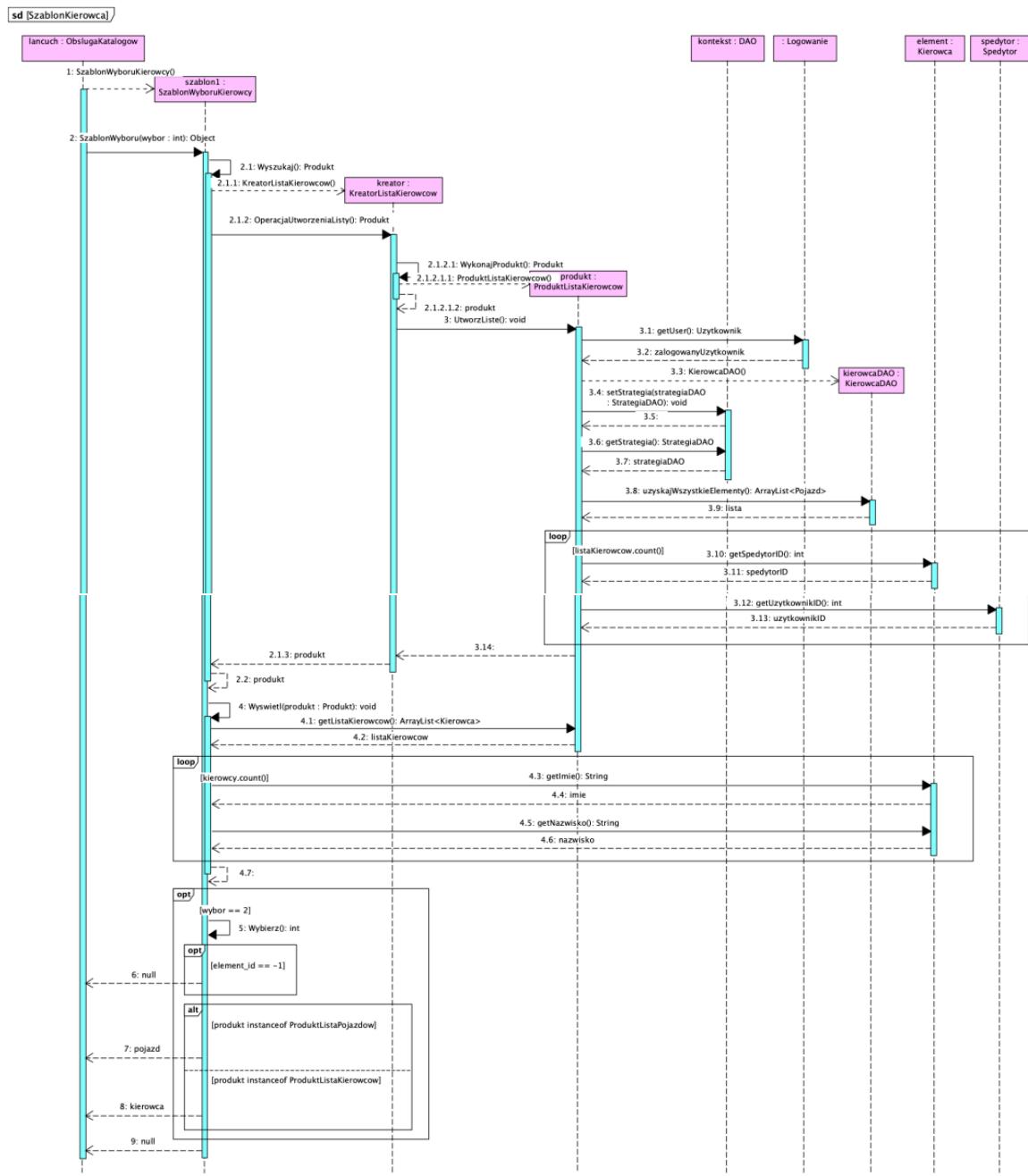
Rysunek 15: Ogólny diagram sekwencji zawierający wszystkie możliwe sekwencje



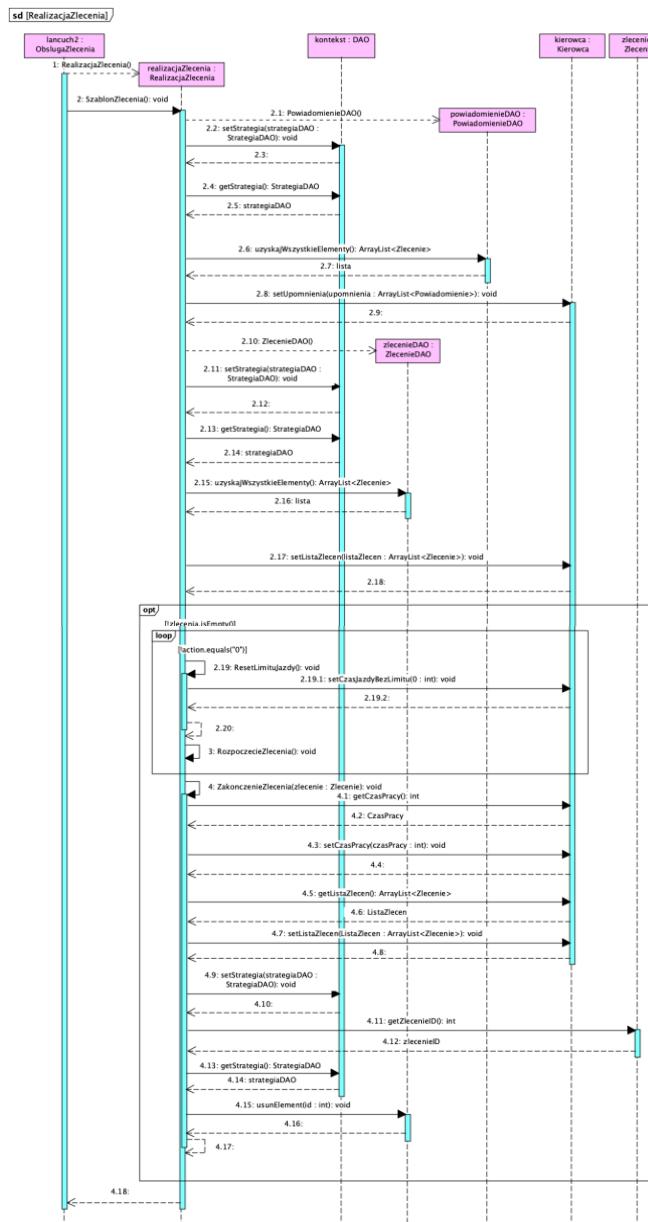
Rysunek 16: Diagram realizacji przydzielania zleceń



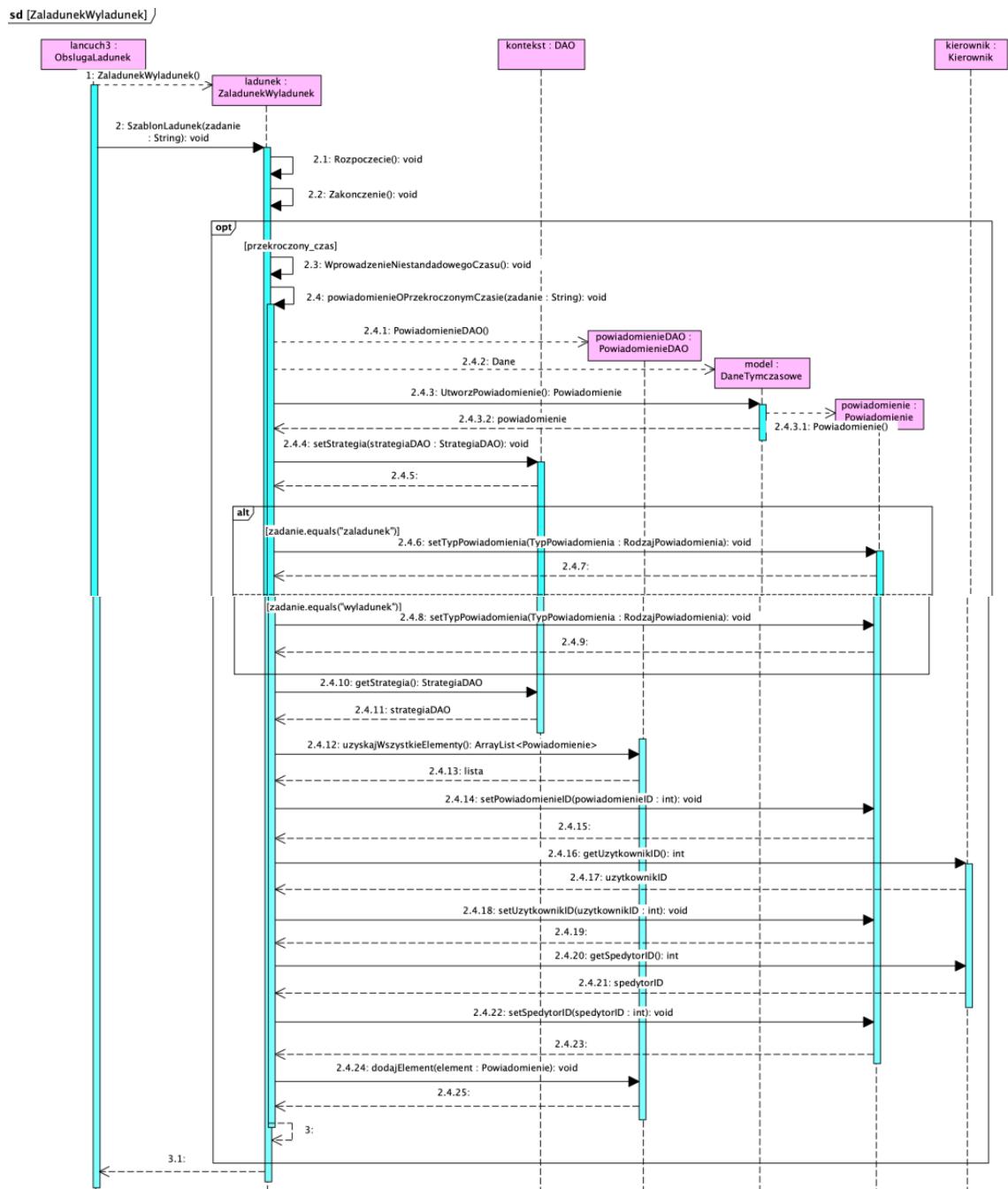
Rysunek 17: Diagram metody szablonowej i wytwórczej dla pojazdów



Rysunek 18: Diagram metody szablonowej i wytwórczej dla kierowcy



Rysunek 19: Diagram Realizacji zlecenia

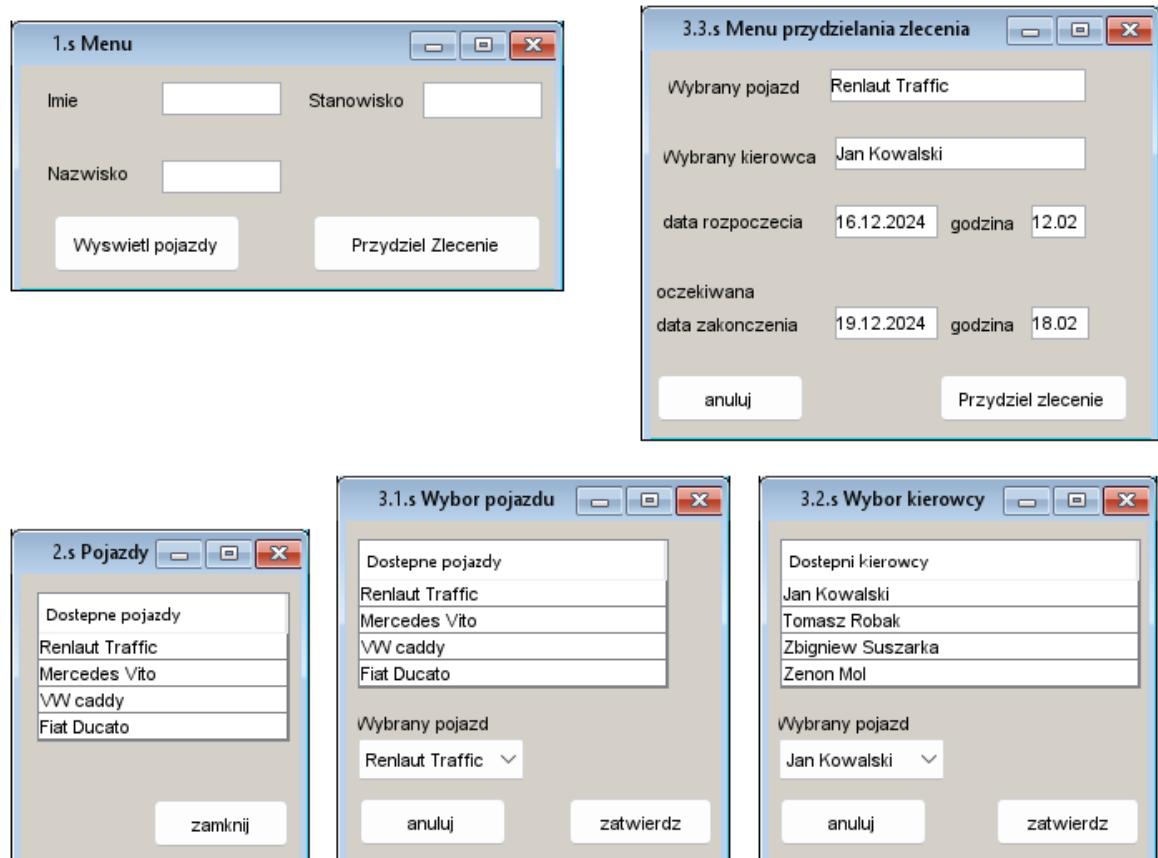


Rysunek 20: Realizacja Załadunku i Wyładunku

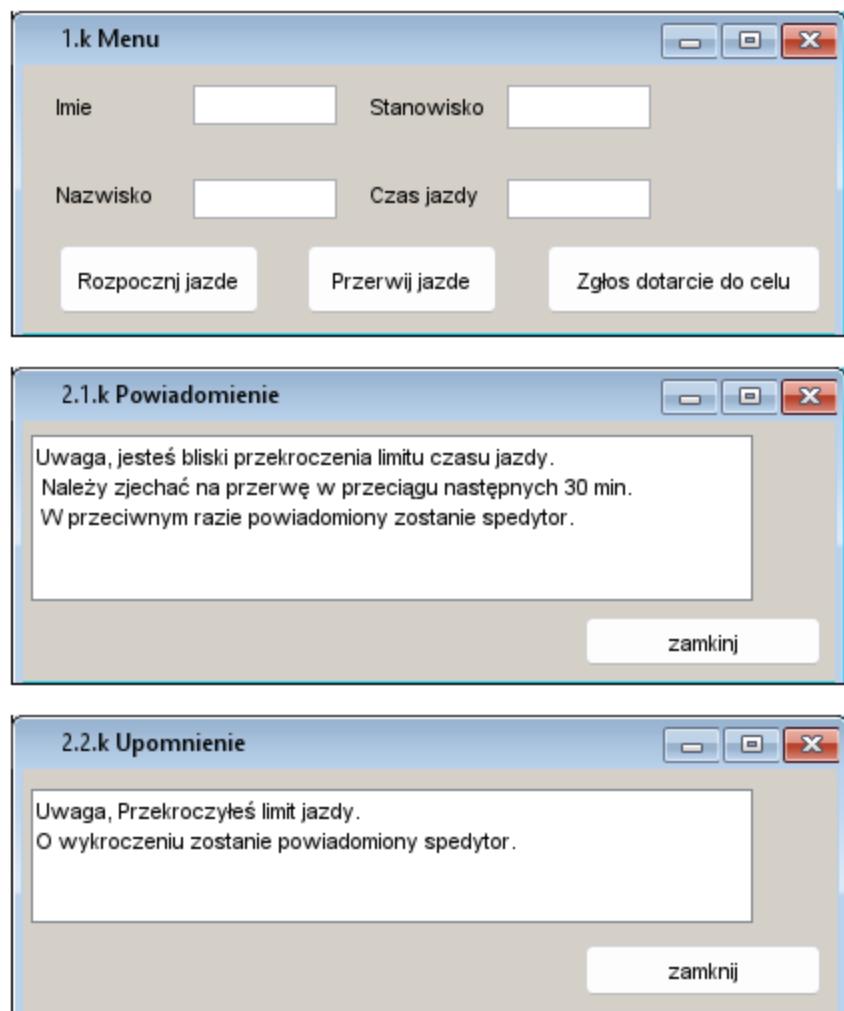
13 Makiety okienkowej warstwy prezentacji



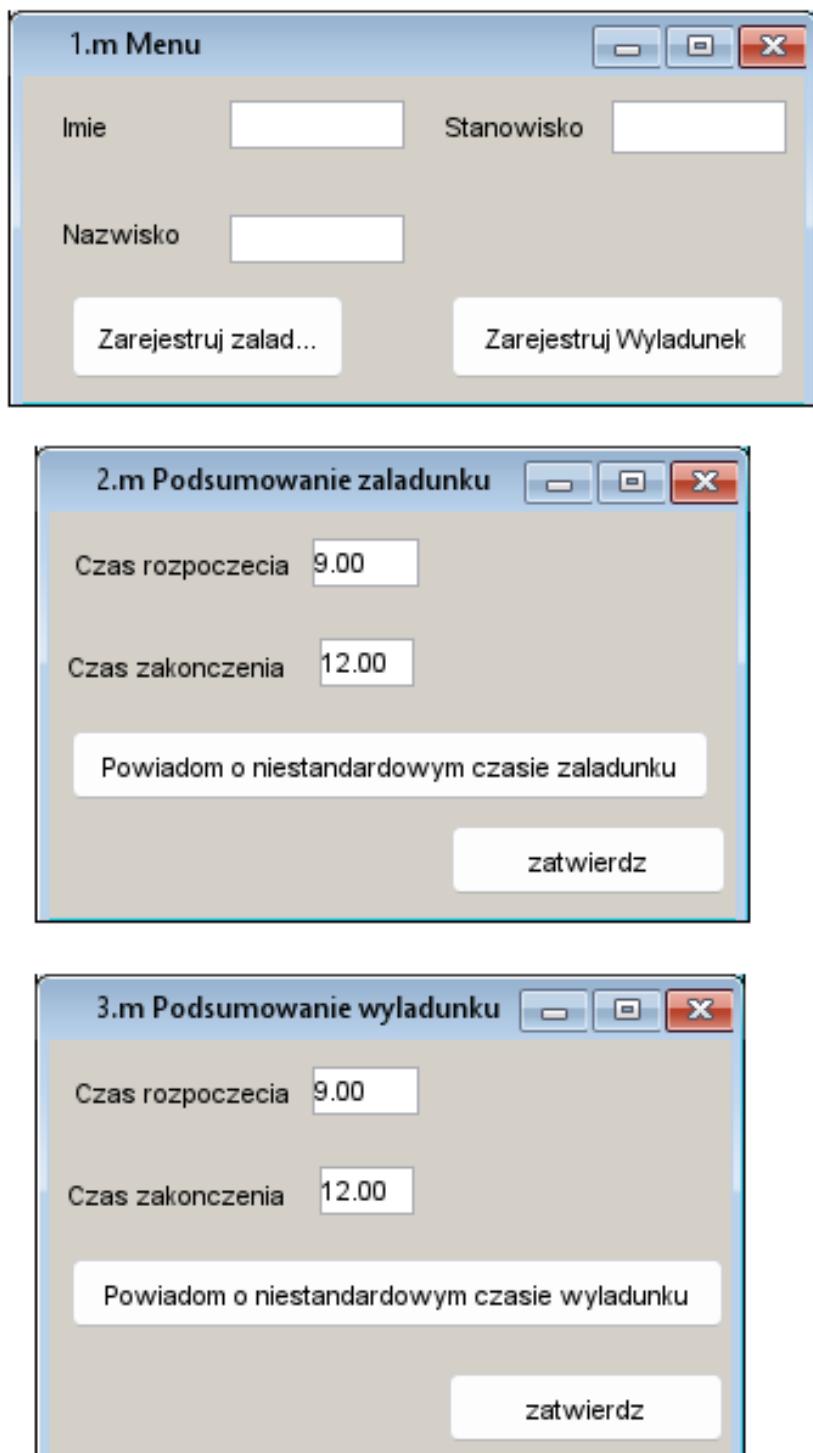
Rysunek 21: Makieta okienkowa logowania



Rysunek 22: Makieta okienkowa warstwy prezentacji pokazująca wygląd ekranu użytkownika oprogramowania, dla stanowiska spedytora.

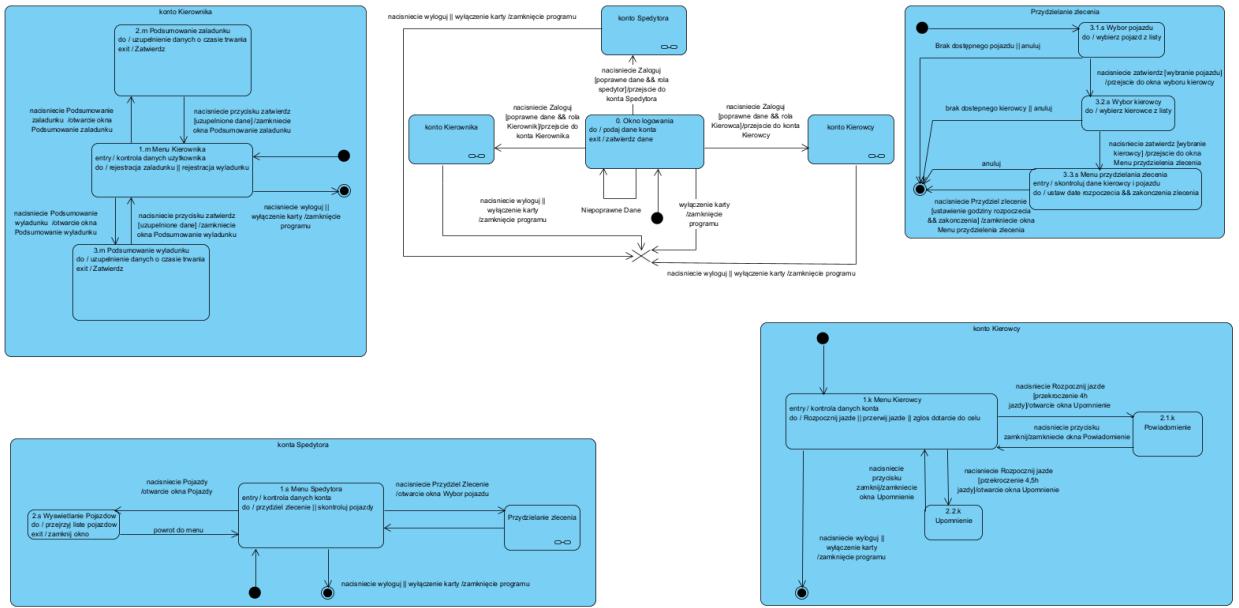


Rysunek 23: Makieta okienkowa warstwy prezentacji pokazująca wygląd ekranu użytkownika oprogramowania, dla stanowiska kierowcy.



Rysunek 24: Makieta okienkowa warstwy prezentacji pokazująca wygląd ekranu użytkownika oprogramowania, dla stanowiska kierownika magazynu.

14 Diagram stanów okienkowej warstwy prezentacji



Rysunek 25: Diagram stanów pokazujący zmienność stanów okienkowej warstwy prezentacji.