

System wspomagający zarządzanie flotą samochodów

Internetowe Bazy Danych - Analiza problemu

Jan PAJDAK
Wojciech SŁOWIŃSKI

6 grudnia 2018

Prowadzący: Dr inż. Roman Ptak
Grupa zajęciowa: TN, Czwartek, 9:15

Spis treści

1	Opis biznesowy	3
2	Wymagania funkcjonalne i нефункционалне	4
2.1	Wymagania funkcjonalne	4
2.2	Wymagania нефункционалне	6
2.2.1	Interfejs użytkownika	6
2.3	Interfejs programistyczny	7
2.4	Bezpieczeństwo	7
3	Przypadku użycia	8
4	SWOT	9
4.1	Powiązania SWOT	10
4.2	Ważność czynników SWOT	10
4.3	Analiza powiązań SWOT	10
4.4	Analiza powiązań TOWS	11
4.5	Wnioski oraz wybór strategii	13
5	Kosztorys	14
6	Harmonogram	15
7	Zastosowane technologie i narzędzia	16
7.1	Zastosowane technologie	16
7.1.1	Interfejs użytkownika	16
7.1.2	Interfejs programistyczny	16
7.2	Wykorzystane narzędzia	17
7.2.1	Repozytorium	17
7.2.2	Edytory i środowiska programistyczne	17
7.2.3	Dodatkowe narzędzia	17

8	Projekt i implementacja	18
8.1	Architektura	18
8.2	Standardy	19
8.3	Interfejs programistyczny	20
8.3.1	Logika biznesowa	20
8.3.2	Struktura solucji	22
8.3.3	Wstrzykiwanie zależności	23
8.4	Sposób działania	23
9	Eksport statystyk floty	25
9.1	Interfejs użytkownika	26
9.1.1	Układ interfejsu użytkownika	26
9.1.2	Walidacja danych	26
9.1.3	Stopka audytowa	28
9.1.4	Dialogi	28
10	Podsumowanie	29
10.1	Wnioski	29
10.2	Możliwości rozwoju	29

1 Opis biznesowy

Produkt skierowany jest do przedsiębiorstw posiadających niewielką flotę samochodów, które nie są przydzielone na stałe do jednego pracownika - pracownik wypożycza samochód, gdy zaistnieje taka potrzeba jak dojazd do klienta lub przejazd między oddziałami firmy. Obecnie większość aplikacji służących do zarządzania flotą dzieli się na dwa rodzaje:

1. *Aplikacje dla firm spedycyjnych* - kluczowym elementem tych systemów jest warstwa pozwalająca na zarządzanie ładunkami oraz zadaniami i trasami; w przypadku grupy docelowej naszego projektu takie mechanizmy są zbędne
2. *Aplikacje dla wypożyczalni samochodów* - systemy te kładą nacisk na mechanizm wypożyczania, ukrywając aspekty techniczne przed użytkownikami; w naszej aplikacji użytkownicy samochodów muszą również zwracać uwagę na parametry takie jak przebieg

Nasz produkt będzie umożliwiał śledzenie zarówno wypożyczeń, jak i stanu pojazdów. Sposób użytkowania samochodów przez pracowników będzie monitorowany, by zapobiec oszustwom takim jak kradzież paliwa. Statystyki na temat floty pozwolą klientom zaoszczędzić pieniądze, przykładowo wskazując drogie w utrzymaniu, nieekonomiczne modele samochodów.

2 Wymagania funkcjonalne i нефункционалне

2.1 Wymagania funkcjonalne

Wymagania zostały opisane według poniższego wzorca:

Numer	Numer wymagania
Nazwa	Krótka nazwa
Opis	Dokładny opis
Aktor	Grupa użytkowników
Kryterium spełnienia	Funkcjonalność, która musi zostać zaimplementowana by wymaganie można było uznać jako spełnione
Ograniczenia	Ograniczenia funkcjonalności, jeżeli takie istnieją

Rozróżniane są dwa rodzaje aktorów:

- Kierowca - użytkownik korzystający z funkcjonalności tworzenia i przeglądania historii rezerwacji.
- Kierownik - użytkownik z pełnym dostępem do systemu.

Kierownik posiada wszelkie prawa i możliwości Kierowcy.

Dodatkowe pojęcia związane z modelami świata biznesowego:

- **Model Pojazdu** — model opisujący specyfikację techniczną wspólną dla pewnego zbioru pojazdów.
- **Pojazd** — model opisujący informacje unikatowe dla pewnego przedstawiciela zbioru Modeli Pojazdów.

Numer	1
Nazwa	Zarządzanie modelami pojazdów
Opis	System powinien pozwalać na dodawanie i edycję modeli pojazdów; specyfikacji technicznej dla danego modelu.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może dodawać nowe modele samochodów. Informacje mogą zostać w późniejszym czasie zmodyfikowane lub usunięte.
Ograniczenia	Model pojazdu może zostać usunięty wyłącznie, gdy nie ma żadnych pojazdów

Numer	2
Nazwa	Zarządzanie pojazdami
Opis	System powinien pozwalać na dodawanie i edycję pojazdów będących egzemplarzami modeli z wymagania #2; pojazd zawiera informacje unikalne dla danego egzemplarza, takie jak numer rejestracyjny.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może dodawać nowe pojazdy dla wybranego modelu. Informacje mogą zostać w późniejszym czasie zmodyfikowane lub usunięte.
Ograniczenia	Pojazd nie może być modyfikowany, gdy jest obecnie zarezerwowany. Pojazd który był rezerwowany, nie może zostać usunięty — może zostać oznaczony jako wycofany z użycia.

Numer	3
Nazwa	Zarządzanie ubezpieczeniami pojazdu
Opis	System powinien umożliwiać wprowadzanie informacji związanych z ubezpieczeniami danego pojazdu.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może przeglądać historię ubezpieczeń danego pojazdu oraz wprowadzać nowe dane. System bierze pod uwagę obecny stan pojazdu podczas tworzenia rezerwacji; pojazd nie może zostać zarezerwowany w okresie gdy nie ma aktywnego ubezpieczenia.
Ograniczenia	

Numer	4
Nazwa	Zarządzanie serwisami pojazdu
Opis	System powinien umożliwiać wprowadzanie informacji związanych z serwisami danego pojazdu.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może przeglądać historię napraw danego pojazdu oraz wprowadzać nowe dane. System rozróżnia różne rodzaje serwisowania takie jak regularny przegląd, zdarzenie wyjątkowe czy naprawa powypadkowa. System bierze pod uwagę obecny stan pojazdu podczas tworzenia rezerwacji; pojazd nie może zostać zarezerwowany, gdy jest obecnie naprawiany.
Ograniczenia	

Numer	5
Nazwa	Tworzenie rezerwacji
Opis	System powinien umożliwiać przeglądanie dostępnych pojazdów (dostępność określana jest na podstawie informacji z wymagania #2) i tworzenie rezerwacji wraz z niezbędnymi danymi takimi jak okres i potrzeba stojąca za rezerwacją
Aktor	Kierowca
Kryterium spełnienia	Kierowca może utworzyć rezerwację
Ograniczenia	Kierowca nie może utworzyć rezerwacji dla innego użytkownika

Numer	6
Nazwa	Kontrola rezerwacji
Opis	System umożliwia kontrolowanie stanu rezerwacji. Rezerwację uznane jest za obowiązującą dopiero po akceptacji przez uprawnioną do tego osobę.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może przeglądać rezerwacji utworzone przez użytkowników systemu oraz zmieniać ich obecny stan po ocenie zasadności rezerwacji
Ograniczenia	Kierownik nie może akceptować własnych rezerwacji

Numer	7
Nazwa	Zbieranie informacji o kosztach rezerwacji
Opis	System umożliwia śledzenie kosztów utrzymania floty na podstawie raportów wprowadzanych przez kierowców.
Aktor	Kierowca
Kryterium spełnienia	Kierowca może wprowadzić informację związane z rezerwacją (zużyte litry paliwa, przejechane kilometry, całkowity koszt) po oddaniu samochodu.
Ograniczenia	

Numer	8
Nazwa	Zbieranie informacji o kosztach utrzymania
Opis	System umożliwia śledzenie kosztów utrzymania floty związanych z ubezpieczeniami oraz naprawami.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może wprowadzić koszty związane z ubezpieczeniem/serwisem pojazdu.
Ograniczenia	

Numer	9
Nazwa	Wyświetlanie informacji o kosztach utrzymania
Opis	System jest w stanie wygenerować plik kompatybilny z programem <i>Excel</i> zawierający dane na temat kosztów floty.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może wywołać utworzenie pliku ze statystykami
Ograniczenia	

Numer	10
Nazwa	Przechowywanie informacji audytowych
Opis	System zapisuje informacje o dacie i użytkowniku dokonującym wprowadzenia nowych danych lub modyfikacji istniejących.
Aktor	Kierownik
Kryterium spełnienia	Informacje o dacie i użytkowniku modyfikowane są w trakcie zapisu do bazy danych. Kierownik może przeglądać dane audytowe.
Ograniczenia	

2.2 Wymagania niefunkcjonalne

2.2.1 Interfejs użytkownika

Wymagania dotyczące wyglądu aplikacji są następujące:

- wygląd powinien być prosty i nowoczesny.
- elementy strony powinny być rozmieszczone w intuicyjny sposób.
- struktura widoków powinna być ułożona zgodnie z zależnościami między wyświetlanymi danymi.
- aplikacja powinna być wygodna w użyciu na ekranach komputerów o rozdzielczości HD (1366x768 pikseli) lub większej.

2.3 Interfejs programistyczny

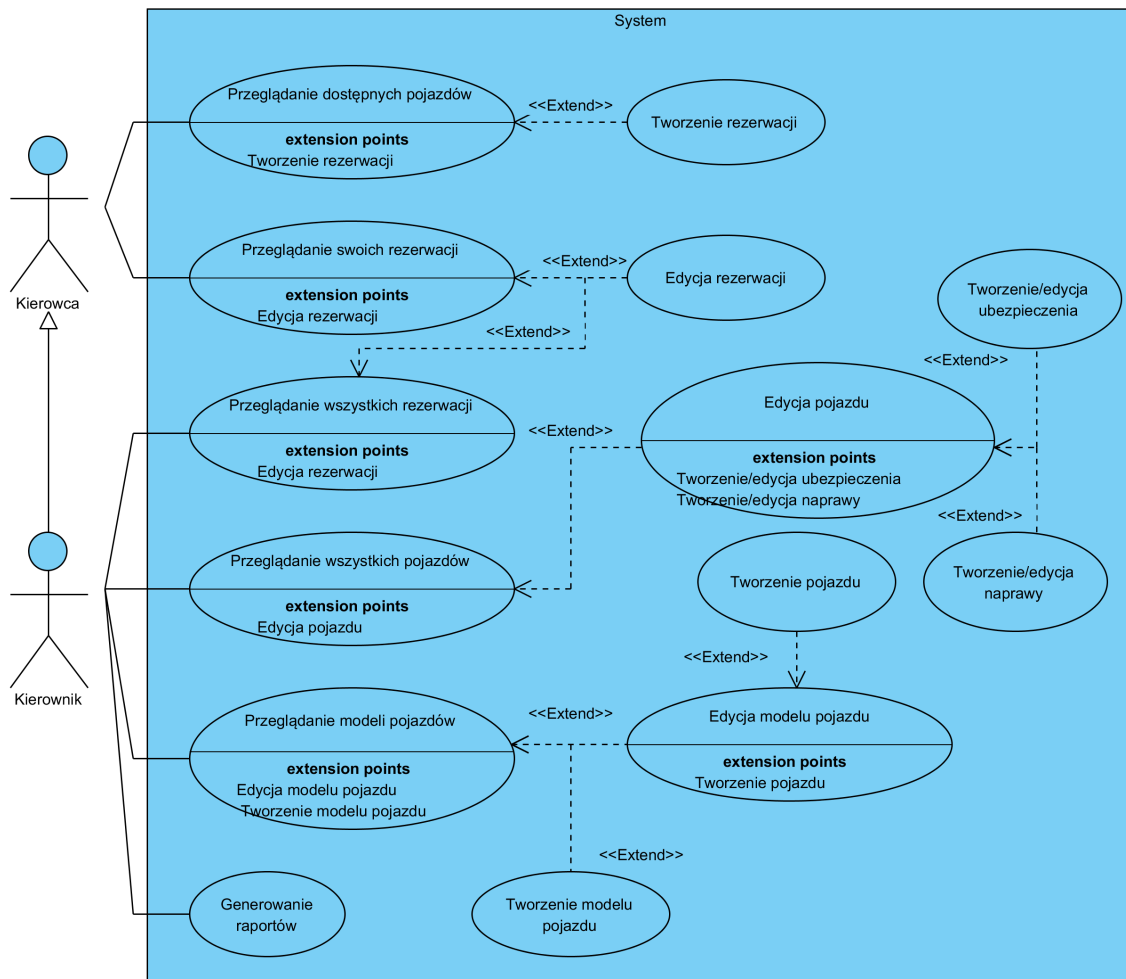
Wymagania dotyczące interfejsu programistycznego są następujące:

- system powinien wymagać niewielkich modyfikacji w przypadku integracji z istniejącymi zasobami firmy (np. baza danych pracowników).
- komunikacja powinna opierać się na otwartych i uniwersalnych standardach, np. dane w postaci *JSON* lub *XML* przesyłane protokołem *HTTP*.
- interfejs programistyczny powinien być niezależny od platformy tak by w przyszłości mógł zostać wykorzystany przez inne aplikacje.

2.4 Bezpieczeństwo

System powinien być zabezpieczony zarówno po stronie interfejsu użytkownika (np. blokada przed przejściem do podstrony) oraz po stronie interfejsu programistycznego (ignorowanie zapytań od nieupoważnionych aplikacji). Zabezpieczenie powinno obsługiwać różne poziomy autoryzacji w zależności od roli użytkownika.

3 Przypadku użycia



4 SWOT

<p>S</p> <ul style="list-style-type: none">• Uwzględnione nowoczesne trendy (klient web)• Produkt skierowany do innych użytkowników niż podane rozwiązania• Nowoczesne metody wytwarzania oprogramowania	<p>W</p> <ul style="list-style-type: none">• Mała znajomość sfery biznesowej• Mało doświadczenia w technologiach• Mało czasu na ukończenie projektu
<p>O</p> <ul style="list-style-type: none">• Zdobyć doświadczenia w najnowszych technologiach• Stworzenie unikatowego produktu na rynku	<p>T</p> <ul style="list-style-type: none">• Potencjalne problemy z nieznanymi technologiami

4.1 Powiązania SWOT

- S - O: otwartość na nowe technologie pozwala na zdobycie doświadczenia
- S - T: nowoczesne metody wytwarzania oprogramowania pozwalają ograniczyć problemy
- W - O: mała ilość czasu może wymusić bardziej konserwatywne podejście
- W - T: słabe strony mogą potęgować zagrożenie

4.2 Ważność czynników SWOT

Waga	Czynniki zewnętrzne	Waga	Czynniki zewnętrzne
1.00	Mocne strony	1.00	Słabe strony
0.20	Uwzględnione nowoczesne trendy (klient web)	0.25	Mała znajomość sfery biznesowej
0.50	Produkt skierowany do innych użytkowników niż podobne rozwiązania	0.45	Mało doświadczenia w technologiach
0.30	Nowoczesne metody wytwarzania oprogramowania	0.30	Mało czasu na ukończenie projektu
1.00	Szanse	1.00	Zagrożenia
0.65	Zdobycie doświadczenia w najnowszych technologiach	1.00	Potencjalne problemy z nieznanymi technologiami
0.35	Stworzenie unikatowego produktu na rynku		

4.3 Analiza powiązań SWOT

Czy określona mocna strona pozwala wykorzystać daną szansę?

Mocne strony/szanse	[O1]	[O2]	[O3]	[O4]	[O5]	[O6]	[O7]	[O8]	[O9]	[O10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[S1]	2	1									0.20	3	0.60	3
[S2]	0	2									0.50	2	1.00	1
[S3]	2	1									0.30	3	0.90	2
[S4]												0	0.00	
[S5]												0	0.00	
[S6]												0	0.00	
[S7]												0	0.00	
[S8]												0	0.00	
[S9]												0	0.00	
[S10]												0	0.00	
Waga	0.65	0.35												
Liczba interakcji	4	4	0	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	2.60	1.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
Ranga	1	2												
Suma interakcji												16		
Suma iloczynów													6.50	

Czy określona mocna zstrona pozwala ograniczyć dane zagrożenie?

Mocne strony/zagrożenia	[T1]	[T2]	[T3]	[T4]	[T5]	[T6]	[T7]	[T8]	[T9]	[T10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[S1]	1										0.20	1	0.20	1
[S2]	0										0.50	0	0.00	3
[S3]	1										0.30	1	0.30	2
[S4]												0	0.00	
[S5]												0	0.00	
[S6]												0	0.00	
[S7]												0	0.00	
[S8]												0	0.00	
[S9]												0	0.00	
[S10]												0	0.00	
Waga	1.00													
Liczba interakcji	2	0	0	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
Ranga	1													
Suma interakcji												4		
Suma iloczynów													2.50	

Czy określona słaba strona ogranicza możliwość wykorzystania danej szansy?

Słabe strony/szanse	[O1]	[O2]	[O3]	[O4]	[O5]	[O6]	[O7]	[O8]	[O9]	[O10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[W1]	0	1									0.25	1	0.25	2
[W2]	1	1									0.45	2	0.90	1
[W3]	2	1									0.30	3	0.90	1
[W4]												0	0.00	
[W5]												0	0.00	
[W6]												0	0.00	
[W7]												0	0.00	
[W8]												0	0.00	
[W9]												0	0.00	
[W10]												0	0.00	
Waga	0.65	0.35												
Liczba interakcji	3	3	0	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	1.95	1.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
Ranga	1	2												
Suma interakcji												12		
Suma iloczynów													5.05	

Czy określona słaba strona potęguje dane zagrożenie?

Słabe strony/zagrożenia	[T1]	[T2]	[T3]	[T4]	[T5]	[T6]	[T7]	[T8]	[T9]	[T10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[W1]	0										0.25	0	0.00	3
[W2]	2										0.45	2	0.90	1
[W3]	1										0.30	1	0.30	2
[W4]												0	0.00	
[W5]												0	0.00	
[W6]												0	0.00	
[W7]												0	0.00	
[W8]												0	0.00	
[W9]												0	0.00	
[W10]												0	0.00	
Waga	1.00													
Liczba interakcji	3	0	0	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
Ranga	1													
Suma interakcji												6		
Suma iloczynów													4.20	

4.4 Analiza powiązań TOWS

Czy określona szansa potęguje daną silną stronę?

Szanse/mocne strony	[S1]	[S2]	[S3]	[S4]	[S5]	[S6]	[S7]	[S8]	[S9]	[S10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[O1]	2	1	2								0.65	5	3.25	1
[O2]	0	2	0								0.35	2	0.7	2
[O3]												0	0	
[O4]												0	0	
[O5]												0	0	
[O6]												0	0	
[O7]												0	0	
[O8]												0	0	
[O9]												0	0	
[O10]												0	0	
Waga	0.20	0.50	0.30											
Liczba interakcji	2	3	2	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	0.4	1.5	0.6	0	0	0	0	0	0	0				
Ranga	3	1	2											
Suma interakcji												14		
Suma iloczynów													6.45	

Czy określone zagrożenie ogranicza daną silną stronę?

Zagrożenia/mocne strony	[S1]	[S2]	[S3]	[S4]	[S5]	[S6]	[S7]	[S8]	[S9]	[S10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[T1]	2	0	2								1.00	4	4	1
[T2]												0	0	
[T3]												0	0	
[T4]												0	0	
[T5]												0	0	
[T6]												0	0	
[T7]												0	0	
[T8]												0	0	
[T9]												0	0	
[T10]												0	0	
Waga	0.20	0.50	0.30											
Liczba interakcji	2	0	2	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	0.4	0	0.6	0	0	0	0	0	0	0				
Ranga	2	3	1											
Suma interakcji												8		
Suma iloczynów													5	

Czy określona szansa pozwala osłabić daną słabą stronę?

Szanse/słabe strony	[W1]	[W2]	[W3]	[W4]	[W5]	[W6]	[W7]	[W8]	[W9]	[W10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[O1]	0	1	0								0.65	1	0.65	1
[O2]	0	0	0								0.35	0	0	2
[O3]												0	0	
[O4]												0	0	
[O5]												0	0	
[O6]												0	0	
[O7]												0	0	
[O8]												0	0	
[O9]												0	0	
[O10]												0	0	
Waga	0.25	0.45	0.30											
Liczba interakcji	0	1	0	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	0	0.45	0	0	0	0	0	0	0	0				
Ranga	2	1	2											
Suma interakcji												2		
Suma iloczynów													1.1	

Czy określone zagrożenie wzmacnia daną słabą stronę?

Zagrożenia/słabe strony	[W1]	[W2]	[W3]	[W4]	[W5]	[W6]	[W7]	[W8]	[W9]	[W10]	Waga	Liczba interakcji	Iloczyn wag i interakcji	Ranga
[T1]	0	2	1								1.00	3	3	1
[T2]												0	0	
[T3]												0	0	
[T4]												0	0	
[T5]												0	0	
[T6]												0	0	
[T7]												0	0	
[T8]												0	0	
[T9]												0	0	
[T10]												0	0	
Waga	0.25	0.45	0.30											
Liczba interakcji	0	2	1	0	0	0	0	0	0	0				
Iloczyn wag i interakcji	0	0.9	0.3	0	0	0	0	0	0	0				
Ranga	3	1	2											
Suma interakcji												6		
Suma iloczynów													4.2	

4.5 Wnioski oraz wybór strategii

Wyniki zbiorcze analizy SWOT/TOWS

Kombinacja	Wyniki analizy SWOT		Wyniki analizy TOWS		Zestawienie zbiorcze SWOT/TOWS	
	Suma interakcji	Suma iloczynów	Suma interakcji	Suma iloczynów	Suma interakcji	Suma iloczynów
Mocne strony [S] / Szanse [O]	16	6.5	14	6.45	30	12.95
Mocne strony [S] / Zagrożenia [T]	4	2.5	8	5	12	7.5
Słabe strony [W] / Szanse [O]	12	5.05	2	1.1	14	6.15
Słabe strony [W] / Zagrożenia [T]	6	4.2	6	4.2	12	8.4

Wyniki analizy strategicznej i wybór strategii

	Szanse	Zagrożenia
Mocne strony	Strategia agresywna	Strategia konserwatywna
	Liczba interakcji 30	Liczba interakcji 12
	Ważona liczba interakcji 12.95	Ważona liczba interakcji 7.5
Słabe strony	Strategia konkurencyjna	Strategia defensywna
	Liczba interakcji 14	Liczba interakcji 12
	Ważona liczba interakcji 6.15	Ważona liczba interakcji 8.4

Dzięki analizie możemy zdecydować się przyjąć strategię agresywną, która opiera się na maksymalnym wykorzystaniu powiązań występujących między silnymi stronami zakładu i szansami, na które możemy trafić.

5 Kosztorys

Koszt projektu prezentuje się następująco:

Opis	Ilość roboczogodzin	Cena za godzinę (netto)	Suma
Zebranie wymagań	24	60	1 440
Stworzenie wstępnego projektu systemu	56	60	3 360
Implementacja	896	80	71 680
Wdrożenia	48	80	3 840
			80 320

Wszystkie ceny są podane w PLN. Jeżeli zajdzie potrzeba dodatkowej pracy na którymkolwiek etapie projektu, koszty będą naliczane według stawek w kolumnie 3.

Koszt wsparcia projektu po wdrożeniu kosztuje 80 PLN / godzina.

Jako platforma pod utworzoną aplikację rekomendowana jest chmura Azure której koszt wynosi 10 000 PLN za rok.

6 Harmonogram

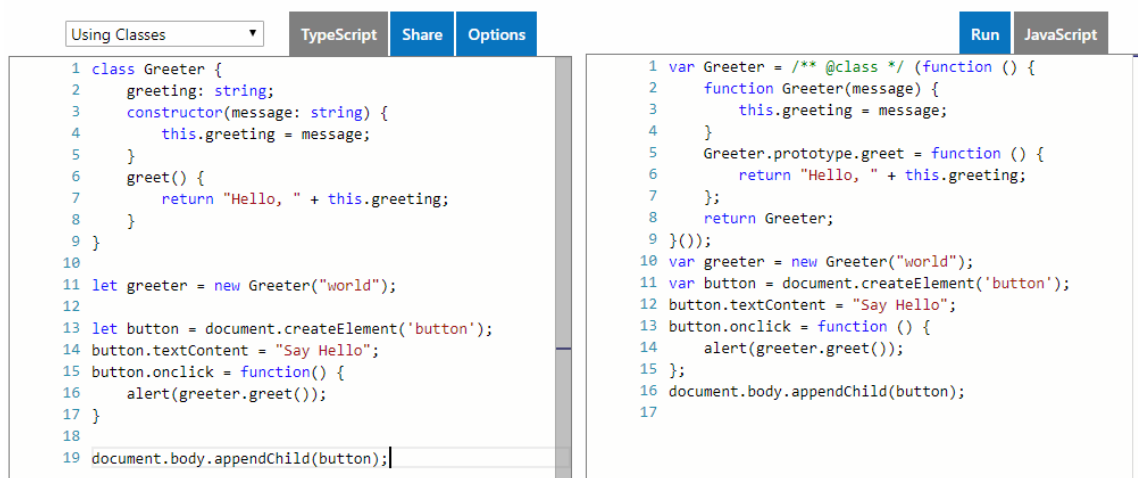
1. Określenie wymagań funkcjonalnych (10.10.2018)
2. Wybór technologii (10.10.2018)
3. Wstępny plan systemu, szkice widoków i bazy danych, diagramy stanów dla obiektów takich jak wypożyczenie (12.10.2018)
4. Stworzenie klas (generacja bazy code-first), tworzenie migracji (16.10.2018)
5. Prace nad strukturą backendu (16.11.2018)
6. Podstawowe operacje REST API (16.11.2018)
7. Testy manualne (Postman) (16.11.2018)
8. Stworzenie aplikacji klienta (widoki kierowcy) (16.11.2018)
9. Zabezpieczenie aplikacji, logowanie użytkownika (20.11.2018)
10. Implementacja warstwy kierownika (backend i frontend) (20.11.2018)
11. Testy automatyczne (25.11.2018)
12. Dalsze usprawnienia systemu (zależne od ilości czasu)

7 Zastosowane technologie i narzędzia

7.1 Zastosowane technologie

7.1.1 Interfejs użytkownika

Interfejs użytkownika wykorzystuje platformę *Angular 7*. Podstawowymi elementami w *Angular* są komponenty Docs [2018a], każdy z nich złożony z: pliku klasy *TypeScript* zawierającej logikę, wzorca *htm* opisującego wygląd widoku oraz opcjonalnego stylu *css*; w przypadku jego braku styl brany jest z komponentu-rodzica. Warto zwrócić uwagę na język programowania wykorzystywany przez platformę *Angular* — *TypeScript* MSDN [2015c], będący rozszerzeniem języka *JavaScript*. *TypeScript* dodaje silniejsze typowanie i kładzie większy nacisk na programowanie obiektowe, jednocześnie pozostając w pełni kompatybilnym z *JavaScript*, do którego jest kompilowany. Proces kompilacji pozwala na usunięcie wielu błędów, które w przypadku *JavaScript* zostałyby zauważone dopiero po uruchomieniu aplikacji.



Rysunek 1: Przykład kompilacji kodu TypeScript do JavaScript. (<http://www.typescriptlang.org>)

Jednym z ważniejszych komponentów aplikacji jest *Bootstrap* - framework interfejsu użytkownika pozwalający w prosty sposób tworzyć estetyczne strony internetowe. Dodatkowo, w projekcie wykorzystano motywy *Bootstrap*.

7.1.2 Interfejs programistyczny

Interfejs programistyczny oparty został na technologii *ASP.NET Core 2.1* — jest to nowoczesna platforma oferująca działanie na wielu systemach operacyjnych oraz większa wydajność względem starszych rozwiązań firmy Microsoft. Wykorzystany język programowania to obiektowy, kompilowany i statycznie typowany *C# 7.3*. Bardzo ważnym elementem tej części projektu jest *EF (Entity Framework) Core 2.1* MSDN [2016a], framework ORM (Object-Relational Mapping) pozwalający na konwersję między tabelami bazy danych a klasami *C#*. Jedną z najważniejszych funkcjonalności *EF Core* jest wykorzystana w niniejszym projekcie możliwość utworzenia bazy danych przy użyciu konwencji *Code First*; baza danych jest automatycznie generowana na podstawie klas *C#* znajdujących się w projekcie. *EF Core* współpracuje z większością popularnych baz danych; na potrzeby tego projektu wykorzystano *MS SQL Server*.

7.2 Wykorzystane narzędzia

W trakcie realizacji projektu wykorzystane zostały narzędzia najczęściej używane przy wybranych technologiach.

7.2.1 Repozytorium

Do zarządzania kodem został wykorzystany system kontroli wersji *Git*. Lokalna kopia projektu była synchronizowana ze zdalnym, prywatnym repozytorium znajdującym się na serwisie GitHub. Wykorzystane rozwiązanie pozwala na łatwy dostęp do wcześniejszych wersji projektu oraz zmniejsza ryzyko utraty kodu, gdyż nie jest on przechowywany tylko w jednym miejscu.

7.2.2 Edytory i środowiska programistyczne

Ze względu na wykorzystane technologie, kod był rozwijany z pomocą narzędzi Microsoft, oferujących najlepsze wsparcie dla *TypeScript* oraz *C#*.

Aplikacja klienta była rozwijana przy użyciu *Visual Studio Code 1.28*, nowoczesnego edytora, który sprawdza się znakomicie przy tworzeniu interfejsów użytkownika ze względu na zintegrowaną konsolę pozwalającą na łatwe zarządzanie paczkami oraz łatwość dostosowywania do potrzeb użytkownika. W trakcie pracy wykorzystano wiele rozszerzeń, najważniejsze z nich to *TSLint*, linter wykrywający błędy w kodzie *TypeScript* oraz *GitLens* — rozszerzenie wspomagające zarządzanie repozytorium *Git*.

Do rozwoju interfejsu programistycznego wykorzystano *Visual Studio 2017* pozwalające na łatwe debugowanie kodu oraz analizę aspektów takich jak wykorzystanie zasobów przez program. *Visual Studio* zostało wzbogacone o narzędzie *JetBrains ReSharper* automatycznie formatujące pliki projektu według zadanego wzorca, zapewniając spójność i przejrzystość kodu.

7.2.3 Dodatkowe narzędzia

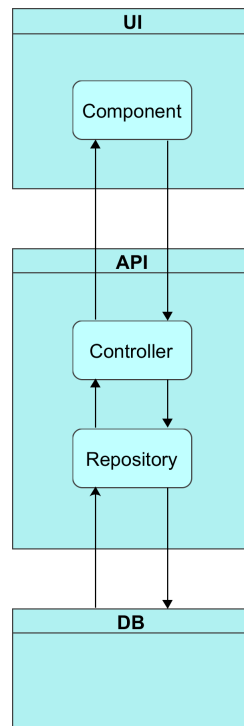
Interfejs programistyczny testowany był przy pomocy *Postman 6.5.2*, aplikacji pozwalającej na wysyłanie oraz zarządzanie zapytaniami HTTP.

Do tworzenia diagramów wykorzystano program *Visual Paradigm*.

8 Projekt i implementacja

8.1 Architektura

System został stworzony przy użyciu klasycznej architektury, w której można wyodrębnić trzy moduły - interfejs użytkownika (*UI*), interfejs programistyczny (*API*) oraz bazę danych (*DB*).



Rysunek 2: Uproszczony schemat architektury z wyodrębnionymi najważniejszymi elementami składowymi.

System został zaprojektowany tak, by mógł zostać zintegrowany z istniejącymi zasobami firmy — jedyne dane, jakie przechowuje, dotyczą logiki biznesowej, związanej z wymaganiami funkcjonalnymi; wynika to z faktu, że większość firm ma już własne bazy danych przechowujące informacje o pracownikach więc duplikacja danych jest niepożądana ze względu na zużycie zasobów oraz możliwe problemy z synchronizacją. Dane związane z użytkownikami (np. imię, nazwisko, e-mail i numer telefonu) czy lokacjami firmy (np. adres) mogą zostać pobrane z innej bazy danych; ponadto interfejs użytkownika nie umożliwia wprowadzania lub edycji takich danych. Implementacja opisana w dalszej części niniejszej pracy przechowuje przykładowe dane użytkowników do celów testowych w tej samej bazie danych, jednakże konfiguracja systemu tak by korzystał z innej, nie stanowi większego problemu.

W architekturze można rozróżnić trzy najważniejsze składowe, dwie pierwsze w interfejsie programistycznym i trzecią w interfejsie użytkownika:

- Kontroler (*Controller*) to klasa odpowiadająca za obsługę żądań *HTTP* MSDN [2018].
- Repozytorium (*Repository*) zawiera logikę pośredniczącą w komunikacji między *API* a bazą danych.
- Komponent (*Component*) to podstawowy element definiujący działanie widoku w *Angular* Docs [2018a].

8.2 Standardy

Projekt był tworzony zgodnie z dobrymi praktykami programowania, z naciskiem na poprawną implementację obiektowego paradygmatu programowania. Interfejs programistyczny był tworzony z użyciem sztandarowych możliwości języka C# takimi jak typy ogólne MSDN [2015b] (*Generics*) pozwalające na tworzenie pojedynczych metod i klas zdolnych do operacji na wielu typach, zachowując wszystkie zalety silnego, statycznego typowania i wysoką wydajność.

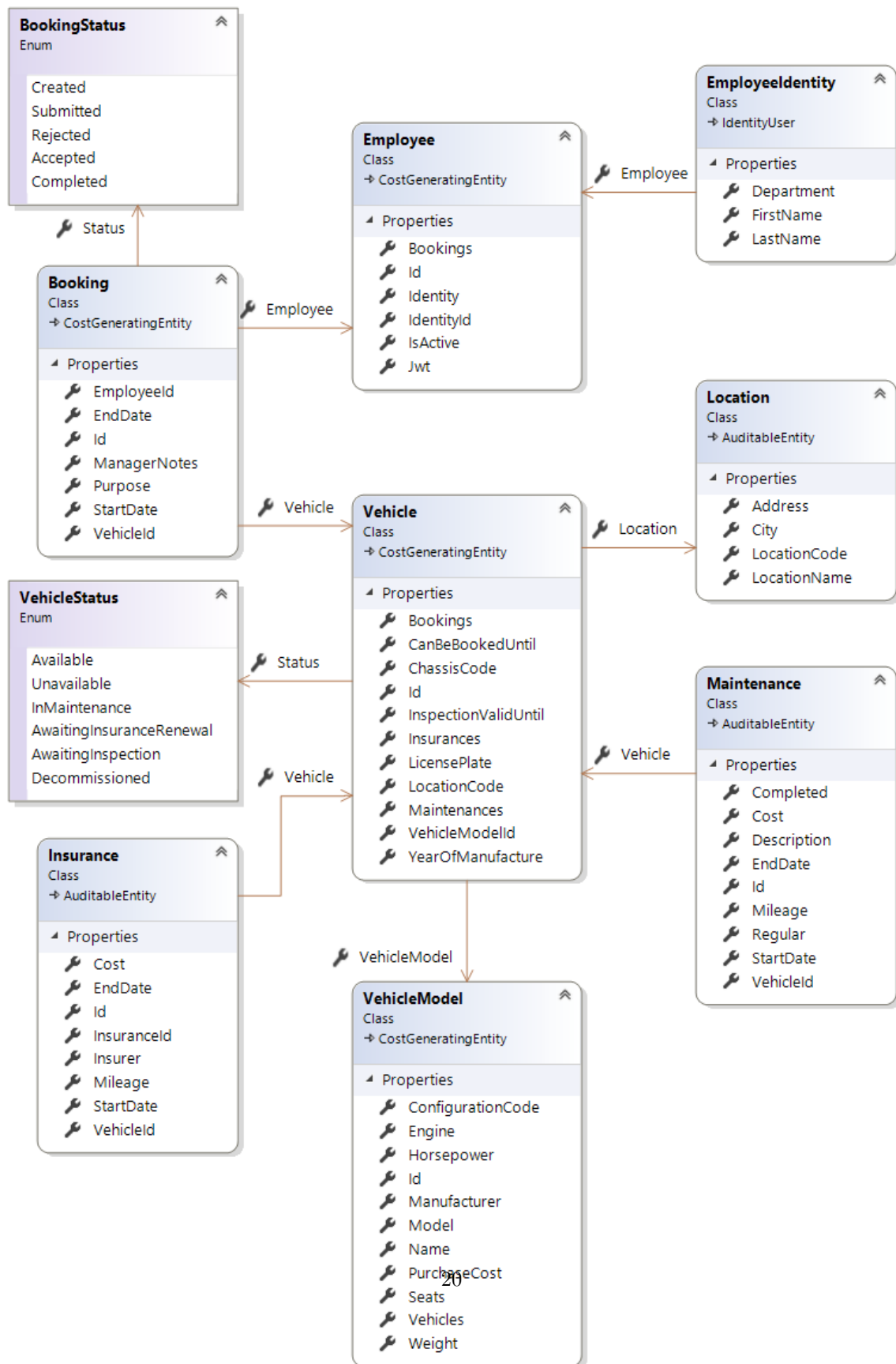
W celu zapewnienia przejrzystości kodu, nazewnictwo wszystkich elementów oraz dokumentacja kodu są zgodne ze standardową konwencją danego języka. Kod jest napisany w całości w języku angielskim.

Język	Typy	Pliki	Zmienne prywatne	Inne zmienne
C# MSDN [2017]	PascalCase	PascalCase.cs	camelCase	PascalCase
TypeScript Docs [2018b]	PascalCase	snake-case.typ.ts	camelCase	camelCase

Tablica 1: Najważniejsze konwencje nazewnicze.

8.3 Interfejs programistyczny

8.3.1 Logika biznesowa



Tablica 2: Klasy obiektów biznesowych.

Klasa	Opis
VehicleModel	Specyfikacja techniczna wspólna dla wielu pojazdów
Vehicle	Informacje unikatowe dla pewnego pojazdu
Insurance	Ubezpieczenie
Maintenance	Naprawa, serwis pojazdu
Employee	Klasa używana do powiązania logiki biznesowej z informacjami o pracowniku
EmployeeIdentity	Dane personalne użytkownika; mogą być pobierane z innej bazy danych
Booking	Rezerwacja pojazdu
Location	Informacje o budynkach należących do firmy korzystającej z systemu; mogą być pobierane z innej bazy danych

Baza danych została automatycznie wygenerowana na podstawie klas opisujących świat biznesowy, przy użyciu *EF Core*.

Do zdefiniowania relacji między tabelami należy użyć pól typu takiego samego jak *PK* docelowej klasy MSDN [2016b]. Używanie adnotacji nie jest wymagane, o ile pole zostało nazwane według standardowej konwencji *EF Core* — *NazwaKlasyId* (np. *VehicleId*). Dodatkowo klasę można uzupełnić o pola nawigacyjne (*navigation properties*), pozwalające na odnoszenie się do powiązanej klasy w łatwy sposób w kodzie programu, należy jednak pamiętać że domyślnie *EF Core 2.1* nie wczytuje informacji o powiązanych obiektach; podczas komunikacji z bazą daną należy jawnie wywołać ładowanie powiązanych obiektów za pomocą metody *LINQ Include()*.

Definiowanie właściwości kolumn wygenerowanych w bazie odbywa się poprzez umieszczenie odpowiednich adnotacji przy polach:

- [Key]: klucz główny (*PK*).
- [Required]: pole jest wymagane, nie może być puste (null).

Listing 1: Przykład definiowania relacji między klasami w code-first.

```
public class Booking
{
    [Key]
    public int Id { get; set; }

    [Required]
    public int VehicleId { get; set; }

    public virtual Vehicle Vehicle { get; set; }
}

public class Vehicle
{
    [Key]
    public int Id { get; set; }

    public virtual ICollection<Booking> Bookings { get; set; }
}
```

Listing 2: Ładowanie powiązanych obiektów na przykładzie relacji rezerwacji do pojazdu.

```
public override Task<Booking> GetById(int id)
{
    return Set
        .Include(b => b.Vehicle)
        .AsNoTracking()
        .SingleOrDefaultAsync(b => b.Id == id);
}
```

Wszystkie klasy związane z logiką biznesową dziedziczą po klasie *AuditableEntity* posiadającej pola przechowujące informacje (data i nazwa użytkownika) o utworzeniu i ostatniej edycji. Klasy związane z elementami generującymi koszty (pojazdami, modelami pojazdów, rezerwacjami i użytkownikami) dodatkowo dziedziczą po klasie *CostGeneratingEntity* przechowującej informacje o koszcie, zużytym paliwie i przejechanych kilometrach.

Wybrana strategia dziedziczenia to *TPC* — *Table per Concrete Type*. W strategii *TPC* tabele utworzone w bazie danych zawierają wszystkie kolumny odpowiadające polom wszystkich klas w hierarchii dziedziczenia.

Listing 3: Klasa abstrakcyjna AuditableEntity.

```
public abstract class AuditableEntity
{
    [Required]
    public DateTime AddedOn { get; set; }

    [Required]
    public string AddedBy { get; set; }

    public DateTime? ModifiedOn { get; set; }

    public string ModifiedBy { get; set; }
}
```

Listing 4: Klasa abstrakcyjna CostGeneratingEntity.

```
public abstract class CostGeneratingEntity : AuditableEntity
{
    [Required]
    public int Mileage { get; set; }

    [Required]
    public int FuelConsumed { get; set; }

    [Required]
    [Column(TableName = "decimal(16,2)")]
    public decimal Cost { get; set; }

    [NotMapped]
    public double AverageFuelConsumption => Mileage == 0
        ? 0
        : (FuelConsumed / Mileage);
}
```

8.3.2 Struktura solucji

Kod interfejsu programistycznego znajduje się w jednej solucji podzielonej na projekty.

- **Vehifleet** — solucja.

Vehifleet.API — konfiguracja systemu oraz kontrolery; projekt startowy.

Vehifleet.API.QueryFilters — filtry używane w żądaniach GET.

Vehifleet.Data.DbAccess — konfiguracja połączenia z bazą danych.

Vehifleet.Data.Dtos — modele transportowe (*Data Transfer Objects*) używane w komunikacji z interfejsem użytkownika.

Vehifleet.Data.Models — modele używane wewnątrz interfejsu programistycznego oraz przy tworzeniu bazy danych.

Vehifleet.Helper — pomocnicze metody rozszerzające MSDN [2015a].

Vehifleet.Repositories — repozytoria odpowiedzialne za interakcje z bazą danych.

Vehifleet.Services.UserService — obsługa logowania użytkowników.

Vehifleet.Services.CsvService — serwis generujący raporty ze statystykami.

8.3.3 Wstrzykiwanie zależności

Obiekty klas z logiką są tworzone przy użyciu wstrzykiwania zależności (*dependency injection*). Większość obiektów jest tworzona dla konkretnego żądania odebranego przez kontroler; po wykonaniu operacji i zwróceniu odpowiedzi obiekt trafia do puli oczekującej na *garbage collector*.

8.4 Sposób działania

Przykładowa ścieżka w serwisie dla żądania listy pojazdów:

```
[ApiController]
[Route("api/vehicles")]
[Authorize(Policy = "RequireEmployeeRole")]
public class VehicleController : ControllerBase
{
    private readonly IGenericRepository<Vehicle, int> vehicleRepository;
    private readonly IGenericRepository<VehicleModel, int>
        vehicleModelRepository;
    private readonly IGenericRepository<Booking, int> bookingRepository;

    public VehicleController(
        IGenericRepository<Vehicle, int> vehicleRepository,
        IGenericRepository<VehicleModel, int> vehicleModelRepository,
        IGenericRepository<Booking, int> bookingRepository)
    {
        this.vehicleRepository = vehicleRepository;
        this.vehicleModelRepository = vehicleModelRepository;
        this.bookingRepository = bookingRepository;
    }

    [HttpGet]
    public async Task<IActionResult> Get([FromQuery] VehicleFilter filter)
    {
        var query = filter.Filter(vehicleRepository.Get());

        var vehicles = await query
            .Include(v => v.VehicleModel)
            .Include(v => v.Location)
            .ToListAsync();
    }
}
```

```

return Ok(Mapper.Map<IEnumerable<VehicleListItemDto>>(vehicles));
}

[ . . . ]

```

Program po otrzymaniu żądania *GET* przekształca parametry zapytania na obiekt *VehicleFilter* a następnie używa go by przefiltrować zapytanie (*query*) które zostanie przesłane do bazy danych:

```

public class VehicleFilter : IQueryFilter<Vehicle>
{
    public IEnumerable<string> Manufacturer { get; set; }
    public IEnumerable<string> Model { get; set; }
    public IEnumerable<string> LocationCode { get; set; }
    public int? MinBookingDays { get; set; }
    public string Status { get; set; }

    public IQueryable<Vehicle> Filter(IQueryable<Vehicle> query)
    {
        if (Manufacturer.NotNullOrEmpty())
        {
            query = query.Where(v => Manufacturer.Any(m => m == v.VehicleModel.
                Manufacturer));
        }

        if (Model.NotNullOrEmpty())
        {
            query = query.Where(v => Model.Any(m => m == v.VehicleModel.Model));
        }

        if (LocationCode.NotNullOrEmpty())
        {
            query = query.Where(v => LocationCode.Any(l => l == v.LocationCode));
        }

        if (MinBookingDays != null && MinBookingDays > 0)
        {
            query = query.Where(v => (v.InspectionValidUntil - DateTime.UtcNow).
                Days > MinBookingDays);
        }

        if (Enum.TryParse(Status, out VehicleStatus status))
        {
            query = query.Where(v => v.Status == status);
        }

        return query;
    }
}

```

Przefiltrowane zapytanie zostaje wykonane przy użyciu metody *ToListAsync()* - w tym momencie obiekty zostają załadowane do pamięci. Następnym krokiem jest konwersja oraz zwrócenie listy *DTO*.

9 Eksport statystyk floty

System przechowuje informacje na temat bieżących kosztów generowanych przez flotę; raporty zawierające te informacje mogą być wyeksportowane do pliku *.csv* by następnie zostać zaimportowane do narzędzia kalkulacyjnego, takiego jak *Microsoft Excel*.

W utworzonym systemie eksport do pliku wywoływany jest przez żądanie *HTTP POST* na adres *api/reports/generate/days*, gdzie *days* to liczba określająca z jak wielu dni wstecz powinny być brane dane dotyczące rezerwacji.

Listing 5: Przykładowy raport ze statystykami pojazdów.

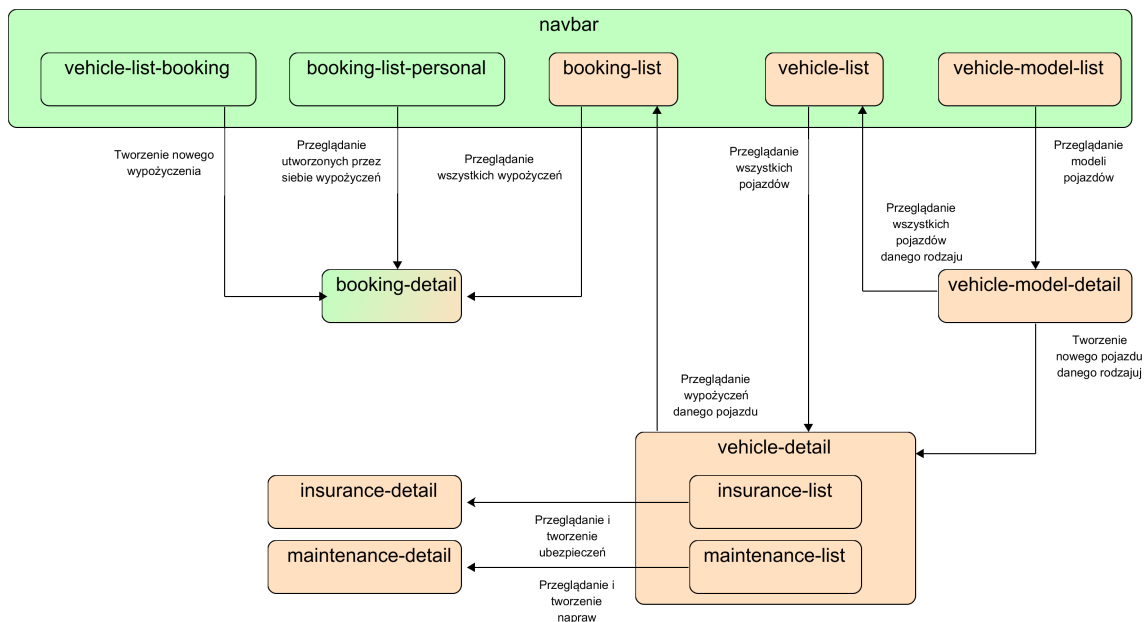
ChassisCode	Manufacturer	Model	YearOfManufacture	Cost	Mileage	FuelConsumed
I1Y9HNIMESHU	Ford	Focus	2016	"9497,00"	7135	373
571VIXS34I71	Ford	C-Max	2016	"4812,00"	9823	532
UTQYXSB44JGE	Toyota	Corolla	2018	"2006,00"	659	0
6T7HDCTIV0BZ	Toyota	Auris	2015	"13743,00"	20026	815
336J4Q7ZF14E	Skoda	Superb	2017	"3166,00"	6474	385
5SX28LAN2LPK	Ford	Focus	2015	"5858,00"	13492	647
MHLI99XWS3OL	Skoda	Octavia	2018	"1023,00"	1465	0
TN2VWMC54JPI	Skoda	Superb	2018	"1194,00"	446	0
TZ1UXM08X7ZU	Ford	Focus	2015	"6823,00"	10776	667
0UYVWETOC5C7	Toyota	Corolla	2017	"4389,00"	5138	224
4VZLW2GQ7JUC	Ford	C-Max	2017	"3802,00"	5781	216
NAJIA1OE0C2B	Ford	Mondeo	2016	"8742,00"	9059	498
HS11MIV1AR8W	Ford	Focus	2017	"4277,00"	8118	362
TIZOCIKJINBR	Skoda	Superb	2018	"3704,00"	47	0
E5RQACK8NWGE	Skoda	Superb	2017	"5618,00"	4630	229

9.1 Interfejs użytkownika

9.1.1 Układ interfejsu użytkownika

Komponenty (widoki) wchodzące w skład interfejsu użytkownika można podzielić na dwa główne rodzaje:

- **widok szczegółowy** (*detail*) który zawiera komplet informacji o danym obiekcie i umożliwia jego edycję.
- **widok listy** (*list*) zawierający małą ilość informacji wymaganych do identyfikacji danego obiektu oraz możliwość przejścia do **widoku szczegółowego**. Widoki tego typu są punktem wejściowym do bardziej zaawansowanej logiki interfejsu, dostępnym bezpośrednio za pomocą paska nawigacji (*navbar*).



Rysunek 4: Widoki interfejsu użytkownika.

Widoki zielone dostępne są dla każdego użytkownika; widoki pomarańczowe wyłącznie dla użytkownika o odpowiednich uprawnieniach. Widok *booking-detail* jest specjalnym przypadkiem oferującym różne możliwości w zależności od uprawnień użytkownika.

9.1.2 Walidacja danych

Dane wprowadzane przez użytkownika są sprawdzane pod kątem poprawności. W przypadku pól tekstowych walidacja najczęściej weryfikuje obecność jakiegokolwiek tekstu; pola, które powinny zawierać liczby, są weryfikowane przy użyciu wyrażeń regularnych.

Listing 6: Przykładowy walidator używający wyrażenia regularnego.

```
mileage: new FormControl('', [  
  Validators.required,  
  Validators.pattern('^[0-9]*$')  
])
```

W przypadku wykrycia błędnych danych, pod polem zawierającym niepoprawne dane pojawia się opis błędu oraz przycisk zatwierdzający zmiany zostaje zablokowany.

Insurance INS-2018-4-10-1021

Insurer:

Insurer name is required.

Insurance ID:

Insurance ID is required.

Start date:

2018-04-10

End date:

2019-04-10

Cost (PLN):

Cost must be a valid number.

Mileage (km):

abcdefghijklm

Mileage must be a valid number.

SaveDelete

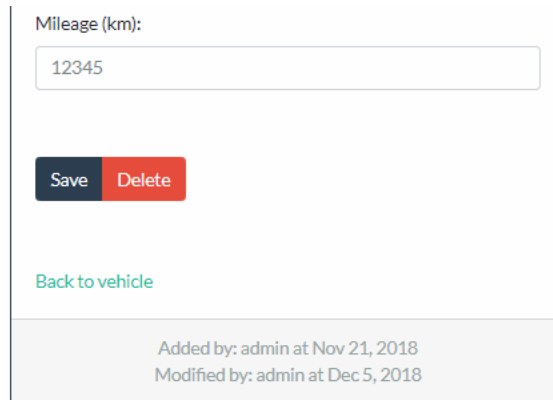
[Back to vehicle](#)

Added by: admin at Nov 21, 2018

Rysunek 5: Widok z polami zawierającymi błędne wartości.

9.1.3 Stopka audytowa

Każdy widok szczegółowy korzysta z komponentu *AuditFooter* wyświetlającego nazwę użytkownika, który utworzył/modyfikował obiekt oraz datę kiedy akcja została wykonana.



Mileage (km):
12345

Save Delete

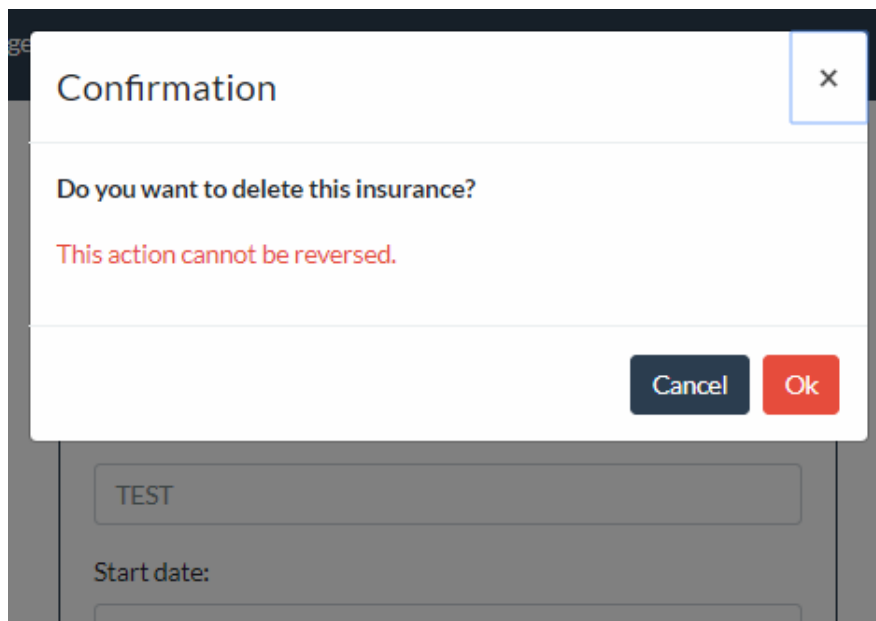
[Back to vehicle](#)

Added by: admin at Nov 21, 2018
Modified by: admin at Dec 5, 2018

Rysunek 6: Stopka audytowa.

9.1.4 Dialogi

Każda akcja która niesie za sobą znaczne (np. akceptacja rezerwacji) lub nieodwracalne (np. usunięcie obiektu) musi zostać zatwierdzona przez użytkownika w dialu który pojawia się po naciśnięciu przycisku zapisu/usuwania.



Confirmation

Do you want to delete this insurance?

This action cannot be reversed.

Cancel Ok

TEST

Start date:

Rysunek 7: Przykładowy dialog.

10 Podsumowanie

10.1 Wnioski

Celem pracy było utworzenie systemu pozwalającego na kontrolę dostępu do pojazdów oraz śledzeniu ich stanu; cel ten został spełniony. System został napisany w sposób zgodny ze standardami, co pozwala na łatwiejsze utrzymanie (w tym dalszą rozbudowę). Zastosowanie nowoczesnych technologii takich jak framework *ASP.NET Core 2.1* pozwala na łatwe rozwijanie aplikacji przy użyciu języka *C#*, dodatkowo oferując wiele zalet takich jak multiplatformowość i zwiększoną wydajność względem tradycyjnego *ASP.NET Framework*. Użycie aplikacji webowej jako interfejsu użytkownika pozwala na dostęp do systemu bez potrzeby instalacji aplikacji klienckiej. Aplikacja webowa eliminuje również problemy takie jak aktualizacje do nowych wersji i zmniejsza koszt utrzymania całego systemu.

10.2 Możliwości rozwoju

System może być rozwinięty na wiele sposobów; kilka z nich:

- automatyczna generacja raportów o kosztach (np. co miesiąc) oraz wprowadzenie serwisu wysyłającego raport e-mailem do użytkowników.
- wyświetlanie statystyk w formie graficznej w interfejsie użytkownika.
- przystosowanie aplikacji to użycia na urządzeniach mobilnych.
- udostępnienie interfejsu programistycznego innym systemom — przykładowo system zbierający koszty generowane przez dany dział w firmie mógłby sprawdzać wydatki pracownika wiążące się z rezerwowaniem pojazdów.
- konteneryzacja aplikacji.
- przechowywanie pełnej historii edycji oraz generowanie raportów audytowych w formacie zgodnym z *Microsoft Excel*.

Literatura

Angular Docs. *Introduction to components*, 2018a. URL <https://angular.io/guide/architecture-components>. Dostęp 03.12.2018.

Angular Docs. *Style Guide*, 2018b. URL <https://angular.io/guide/styleguide>. Dostęp 03.12.2018.

MSDN. *Extension methods*, 2015a. URL <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods>. Dostęp 05.12.2018.

MSDN. *Generics*, 2015b. URL <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/>. Dostęp 03.12.2018.

MSDN. *TypeScript - Understanding TypeScript*, 2015c. URL <https://msdn.microsoft.com/en-us/magazine/dn890374.aspx>. Dostęp 03.12.2018.

MSDN. *Entity Framework Core*, 2016a. URL <https://docs.microsoft.com/en-us/ef/core>. Dostęp 03.12.2018.

MSDN. *Entity Framework Core: Relationships*, 2016b. URL <https://docs.microsoft.com/en-us/ef/core/modeling/relationships>. Dostęp 03.12.2018.

MSDN. *General Naming Conventions*, 2017. URL <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/general-naming-conventions>. Dostęp 03.12.2018.

MSDN. *Build web APIs with ASP.NET Core*, 2018. URL <https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-2.1>. Dostęp 03.12.2018.