

Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej



System informacji oraz sprzedaży biletów
komunikacji miejskiej i międzymiastowej

Bartłomiej Dach, Tymon Felski

Wersja 1.0

13 listopada 2016

Lista zmian w dokumencie:

Data	Autor	Opis zmian	Wersja
16.10.2016	Bartłomiej Dach, Tymon Felski	Określenie wymagań projektu oraz harmonogramu prac	1.0

Spis treści

1 Specyfikacja

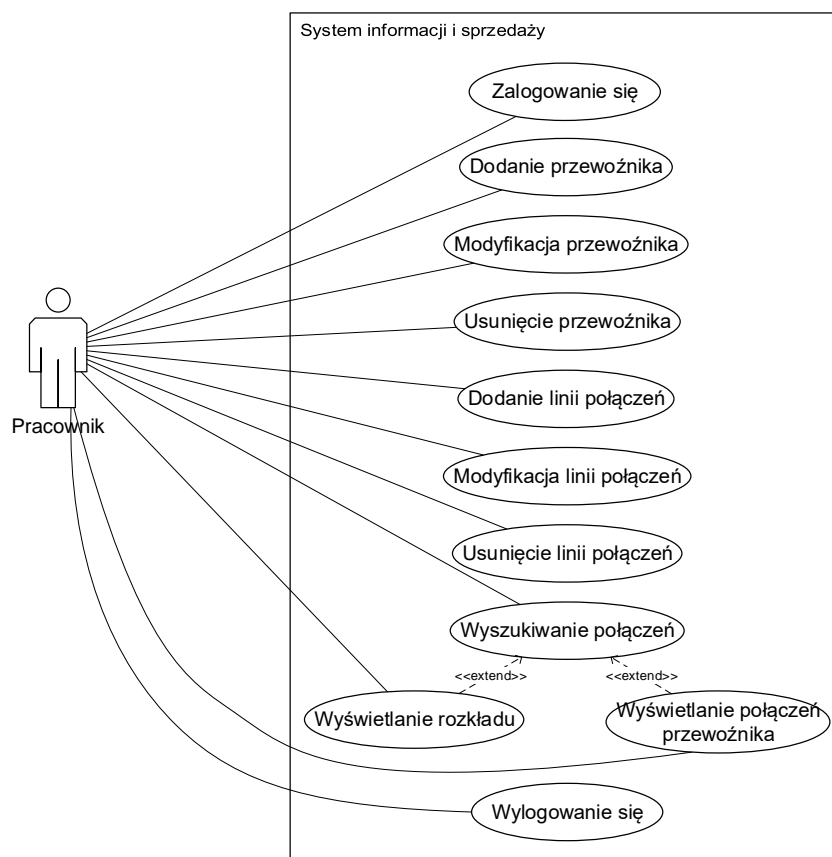
1.1 Opis biznesowy

Niniejszy system służy do przechowywania danych o przewoźnikach i połączeniach komunikacji miejskiej oraz międzymiastowej. Składowane dane wykorzystywane są do wyszukiwania konkretnych połączeń oraz sprzedaży biletów.

1.2 Wymagania funkcjonalne

Przypadki użycia

Poniższy diagram UML przedstawia zbiór przypadków użycia aplikacji dla aktora – pracownika firmy pośredniczącej w sprzedaży biletów wielu przewoźników.



Rysunek 1: Diagram przypadków użycia dla aplikacji

Poszczególne przypadki są opisane szerzej w poniższej tabeli:

Aktor	Nazwa	Opis	Odpowiedź systemu
Pracownik	Zalogowanie się	Zalogowanie się użytkownika do systemu	Potwierdzenie zalogowania się lub komunikat o błędzie
	Dodanie przewoźnika	Dodanie informacji o nowym przewoźniku do bazy	Potwierdzenie dodania danych do bazy
	Modyfikacja przewoźnika	Zmiana danych przewoźnika przechowywanych w bazie	Potwierdzenie zmodyfikowania rekordu
	Dodanie linii połączeń	Dodanie nowej linii połączeń danego przewoźnika	Potwierdzenie dodania linii do bazy
	Modyfikacja linii połączeń	Modyfikacja linii połączeń danego przewoźnika	Potwierdzenie modyfikacji rekordu
	Wyszukiwanie połączeń	Wyszukiwanie połączeń przewoźników między wybranymi punktami	Widok z listą znalezionych połączeń
	Wyświetlanie rozkładu	Wyświetlanie rozkładu jazdy wybranej linii	Widok zawierający informacje o przejazdach na wybranej linii
	Wyświetlanie połączeń przewoźnika	Wyświetlanie połączeń obsługiwanych przez danego przewoźnika	Widok zawierający informacje o liniach danej firmy
	Wylogowanie się	Wylogowanie się pracownika z systemu	Potwierdzenie zakończenia pracy z systemem

Tablica 1: Opisy przypadków użycia dla użytkownika

User stories

1. Interfejs administracyjny dla pracownika

- 1.1. Jako zalogowany pracownik dodaję/modyfikuję przewoźnika.
Dowolny zalogowany pracownik może dodać nowego przewoźnika lub zmodyfikować informacje o przewoźniku, takie, jak: nazwę i adres firmy, numer REGON, dane kontaktowe osoby odpowiadającej za współpracę w kwestii sprzedaży biletów.
- 1.2. Jako zalogowany pracownik dodaję/modyfikuję linię połączeń.
Dowolny zalogowany pracownik może dodać lub zmodyfikować informacje o nowym połączeniu takie jak: przystanki początkowe, pośrednie i końcowe, czas odjazdu i przyjazdu na poszczególnych przystankach, ilość dostępnych miejsc w danym kursie, podstawowa cena biletu oraz wysokość zniżek.
- 1.3. Jako zalogowany pracownik wyświetlam rozkład jazdy dla wybranej linii.
Dowolny zalogowany pracownik może wyświetlić rozkład jazdy wybranej linii. Z poziomu wyświetlania rozkładu pracownik może: dodać nowe połączenie tej linii, usunąć istniejące połączenie lub wygenerować rozkład jazdy w postaci pliku.
- 1.4. Jako zalogowany pracownik wyszukuję połączenie.
Dowolny zalogowany pracownik może wyszukać dostępne połączenia pomiędzy wprowadzonymi miastami.
- 1.5. Jako zalogowany pracownik wyświetlam rozkład jazdy danej linii.
Dowolny zalogowany pracownik może wyszukać rozkład jazdy dla danej linii komunikacyjnej i go wyświetlić.
- 1.6. Jako zalogowany pracownik wyświetlam połączenia dla danego przewoźnika.
Dowolny zalogowany pracownik może wyświetlić połączenia od danego przewoźnika.

1.3 Wymagania нефункциональные












Poniższa tabela zawiera rozpisane wymagania нефункциональные narzucone dla systemu.

Obszar wymagań	Nr	Opis
Użyteczność (<i>Usability</i>)	1	Rozmiar czcionki użytej w aplikacji musi być nie mniejszy niż 12 punktów.
	2	Aplikacja powinna obsługiwać zmianę rozmiaru okna w sposób który umożliwia korzystanie ze wszystkich jej funkcjonalności (tzw. responsive design).
Niezawodność (<i>Reliability</i>)	3	Aplikacja musi być odporna na dokonywanie jednoczesnych zmian tego samego rekordu bazy przez wielu pracowników jednocześnie.
Wydajność (<i>Performance</i>)	4	Aplikacja powinna dodawać nowe obiekty do systemu w czasie nie dłuższym niż 1 sekundę, przy 50 żądaniach dodania obiektu na minutę.
	5	Zużycie pamięci RAM przez aplikację nie powinno przekroczyć 200 megabajtów.
	6	Wyszukiwanie połączenia między określonymi miastami powinno trwać mniej niż 2 sekundy, przy ok. 10 tys. rekordów.
Utrzymanie (<i>Supportability</i>)	7	Do aplikacji dołączona zostanie instrukcja wykonywania kopii zapasowej danych.

Tablica 2: Tabela wymagań niefunkcjonalnych

1.4 Harmonogram projektu

Prace przy projekcie będą realizowane według następującego harmonogramu:

ID	Nazwa zadania	Początek	Koniec	Czas trwania	paź 2016		lis 2016	
					16.10	23.10	30.10	6.11
1	Analiza wymagań projektu	15.10.2016	18.10.2016	4d				
2	Projekt architektury aplikacji	19.10.2016	22.10.2016	4d				
3	Wstępna implementacja	23.10.2016	25.10.2016	3d				
4	Właściwa implementacja	26.10.2016	08.11.2016	14d				
5	Utworzenie encji i serwisów	26.10.2016	29.10.2016	4d				
6	Utworzenie głównego widoku	30.10.2016	01.11.2016	3d				
7	Utworzenie widoków przewoźników i linii	02.11.2016	05.11.2016	4d				
8	Utworzenie widoków wyszukiwania	06.11.2016	08.11.2016	3d				
9	Końcowa dokumentacja, testy	09.11.2016	12.11.2016	4d				
10	Poprawa błędów	13.11.2016	14.11.2016	2d				
11	Zdanie projektu	15.11.2016	15.11.2016	0d				

Rysunek 2: Diagram Gantta z planowanym harmonogramem projektu

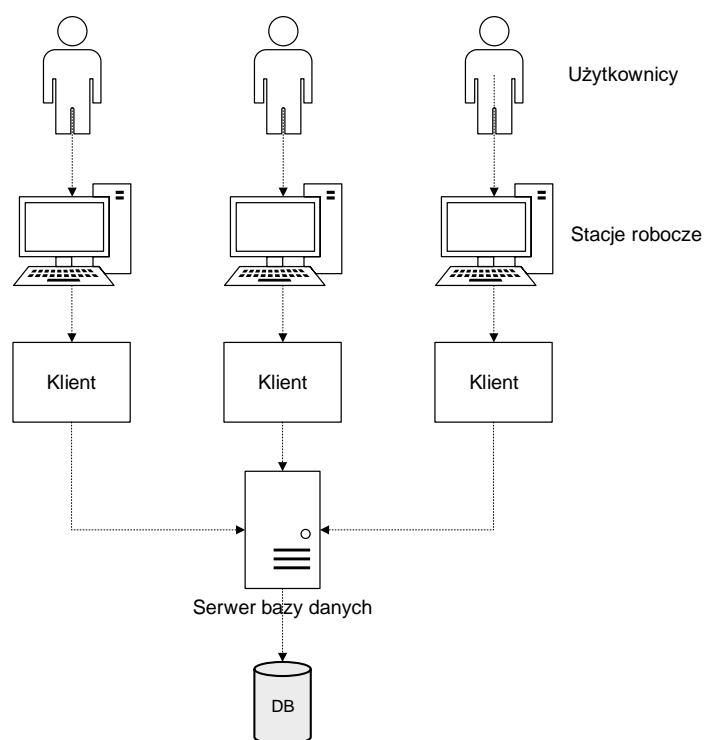
Kamienie milowe:

- 18 października: Zakończenie analizy wymagań funkcjonalnych i niefunkcjonalnych projektu.
- 22 października: Zakończenie projektu architektury aplikacji, łącznie z wyróżnieniem komponentów oraz podsystemów.

3. 25 października: Wstępna implementacja projektu architektury, naniesienie ewentualnych poprawek do architektury wynikających z problemów implementacyjnych.
4. 29 października: Utworzenie encji biznesowych oraz serwisów wykorzystywanych przez użytkowników.
5. 1 listopada: Utworzenie głównego widoku aplikacji.
6. 5 listopada: Utworzenie widoków dodawania przewoźników oraz linii.
7. 8 listopada: Utworzenie widoków wyszukiwania połączeń oraz wyświetlania połączeń danej linii oraz przewoźnika.
8. 12 listopada: Zakończenie dokumentacji, testów aplikacji oraz identyfikacji błędów.
9. 15 listopada: Zakończenie poprawy znalezionych błędów, zdanie projektu łącznie z pełną dokumentacją.

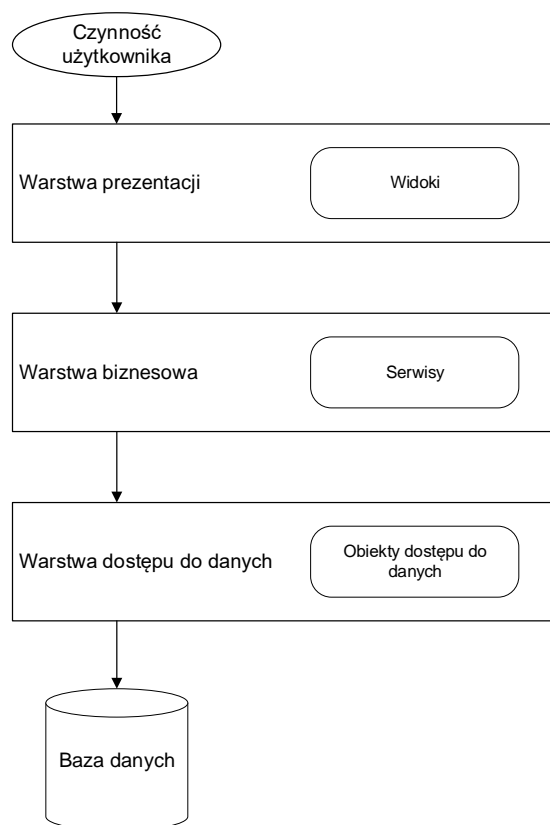
1.5 Architektura rozwiązania

Docelowym środowiskiem aplikacji są małe lub średnie firmy pośredniczące w sprzedaży biletów komunikacyjnych, tzn. przedsiębiorstwa zatrudniające do 250 pracowników, z czego dostęp do systemu miałby dość niski procent tej liczby (w założeniach ok. 20-30%). Dane, których przechowywanie jest niezbędne do spełnienia wymagań funkcjonalnych mają dość małą zmienność - stosunkowo rzadko ulegają zmianom lub przedawnieniom. Dodatkowo, ze względu na wewnętrzny charakter przechowywanych danych, system powinien być scentralizowany i znajdować się w jednym fizycznym położeniu.



Rysunek 3: Schemat architektury systemu

Biorąc pod uwagę opisany powyżej charakter zamówionego rozwiązania, wybrana została prosta architektura z centralną bazą danych oraz aplikacją typu "gruby klient", wykorzystującą bezpośrednie połączenie z bazą. Rozwiązanie to jest spójne z opisanymi cechami systemu, a poza tym jest dość proste we wdrożeniu i nie wprowadza niepotrzebnych kosztów rozproszenia.



Rysunek 4: Schemat architektury aplikacji klienckiej

Planowana architektura aplikacji klienckiej ma charakter warstwowy. Wyróżnione zostały następujące warstwy:

- warstwa dostępu do danych - odpowiedzialna za kontakt z bazą oraz odczyt i zapis przechowywanych tam danych,
- warstwa biznesowa - odpowiedzialna za wykonywanie poszczególnych usług (np. dodania czy modyfikacji przewoźnika),
- warstwa prezentacji - odpowiedzialna za wyświetlanie interfejsu użytkownika.

Głównymi powodami zaproponowania architektury warstwowej były:

- możliwość wymiany silnika bazodanowego oraz warstwy prezentacji bez naruszania warstwy biznesowej,
- podział odpowiedzialności na poszczególne warstwy,
- spójny charakter wymagań - podział na podsystemy jest zbędny.

Ze względu na małą liczbę użytkowników niska skalowalność oraz wydajność rozwiązań warstwowych zostały uznane za ryzyko drugorzędne.

2 Dokumentacja końcowa (powykonawcza)

2.1 Wymagania systemowe

2.2 Biblioteki wraz z określeniem licencji

W budowie aplikacji zostały użyte następujące biblioteki oraz komponenty firm trzecich:

Nr	Komponent i wersja	Opis	Licencja	
1	Castle.Core, 3.3.3	Wykorzystywana do tworzenia obiektów <i>proxy</i> . Zależność biblioteki Moq.	Apache License 2.0	[?]
2	Entity Framework, 6.1.3	Framework do mapowania obiektowo-relacyjnego (ORM).	Apache License 2.0	[?]
3	FluentAssertions, 4.17.0	Wykorzystywany w testach jednostkowych w celu ułatwienia pisania asercji.	Apache License 2.0	[?]
4	Moq, 4.5.28	Używany w testach jednostkowych do tworzenia obiektów zastępczych (tzw. <i>mock object</i>).	BSD 3-Clause	[?]
5	NUnit, 3.5.0	Framework do wykonywania testów jednostkowych.	MIT	[?]
6	ReactiveUI, 6.5.2	Biblioteka wspomagająca w realizacji wzorca MVVM w aplikacji klienckiej, zintegrowana z Reactive Extensions.	MS-PL	[?]
7	Reactive Extensions, 2.2.5	Biblioteka wspomagająca w programowaniu aplikacji opartych na asynchronicznym przetwarzaniu danych oraz zdarzeniach. Zależność ReactiveUI.	Apache License 2.0	[?]
8	Splat, 2.0.0	Kontener IoC wspomagający w realizacji wzorca wstrzykiwania zależności.	MIT	[?]

Tablica 3: Lista użytych bibliotek i komponentów

2.3 Instrukcja instalacji

2.4 Instrukcja uruchomienia

2.5 Instrukcja użycia

2.6 Instrukcja utrzymania

2.7 Raport odstępstw od specyfikacji wymagań

2.8 Dokumentacja usług Web Services

3 Dokumentacja końcowa (powykonawcza) – punkty wymagane przez prowadzącego zajęcia

3.1 Pseudokod

3.2 Diagramy sekwencji

3.3 Model danych

3.4 Scenariusz testów akceptacyjnych

3.5 Raport z przeprowadzonych testów

4 Lista użytych skrótów

BSD Berkeley Software Distribution

MIT Massachusetts Institute of Technology

MS-PL Microsoft Software Public License

MVVM *ang.* Model-View-ViewModel – wzorec używany w projektach realizowanych w technologii WPF pozwalający na odseparowanie logiki aplikacji od warstwy prezentacyjnej.

WPF Windows Presentation Framework

5 Bibliografia

- [1] Castle Project, *Castle Core*, <https://github.com/castleproject/Core>
- [2] ASP.NET, *Entity Framework 6*, <https://github.com/aspnet/EntityFramework6>
- [3] Dennis Doomen, *FluentAssertions*, <https://github.com/dennisdoomen/fluentassertions>
- [4] Moq, *Moq 4*, <https://github.com/moq/moq4>
- [5] NUnit, *NUnit*, <https://github.com/nunit/nunit>
- [6] ReactiveUI, *ReactiveUI*, <https://github.com/reactiveui/ReactiveUI>
- [7] Reactive Extensions, *Rx.NET*, <https://github.com/Reactive-Extensions/Rx.NET>
- [8] Paul Betts, *Splat*, <https://github.com/paulcbetts/splat>