DamberPol

POLITECHNIKA KOSZALIŃSKA

WYDZIAŁ ELEKTRONIKI I INFORMATYKI

STUDIA STACJONARNE, SEMESTR VI, KIERUNEK INFORMATYKA

ROK AKADEMICKI 2015 / 2016

DOKUMENTACJA Z PRZEDMIOTU „PROJEKT ZESPOŁOWY CASE”

TEMAT PROJEKTU

APLIKACJA BAZODANOWA DO OBSŁUGI FIRMY TRANSPORTOWEJ

wykonawcy
Damian Berda
Jakub Wachowiak
Łukasz Stolarczyk
Marcin Chruścicki
Michał Motykowski

Spis Treści

1. **Protokół z zebrania założycielskiego zespołu programistycznego „aplikacja bazodanowa do obsługi firmy transportowej" zwanego dalej zespołem, odbytego w dniu 01.03.2016 r. w Koszalinie.**

W zebraniu wzięły udział następujące osoby:

1. Damian Berda.
2. Jakub Wachowiak.
3. Łukasz Stolarczyk.
4. Marcin Chruścicki.
5. Michał Motykowski.

Uczestnicy zebrania podjęli indywidualną decyzję o przystąpieniu do zespołu programistycznego „aplikacja bazodanowa do obsługi firmy transportowej" zwanego dalej zespołem. Po zapisaniu obecności wszystkich członków oraz wypełnieniu deklaracji członkowskich zespołu przystąpiono do wyboru kierownika zespołu. Podczas głosowania jednogłośnie wybrany został Marcin Chruścicki. Zebrani upoważnili nowo wybranego przedstawiciela do reprezentowania interesów zespołu. Po wybraniu przedstawiciela przystąpiono do ustalenia postanowień ogólnych.

Postanowienia ogólne:

1. Odwołanie przewodniczącego może odbyć się po zgłoszeniu tzw. wotum nieufności względem wyżej wymienionej osoby oraz zatwierdzenie usunięcia poprzez demokratyczne głosowanie.
2. Każdy członek zespołu zobowiązany jest do wykonywania zadań projektowych określonych w harmonogramie zadań.
3. Każdy członek zespołu zobowiązany jest do przestrzegania postanowień ogólnych.
4. Data zakończenia projektu została wstępnie ustalona na dzień 7 VI 2016 r.

Następnie przystąpiono do określenia celów projektu, jak również obowiązków członków zespołu.

Cele projektu:

1. Zaliczenie przedmiotu „Projekt Zespołowy CASE".
2. Wykonanie projektu oprogramowania, które posłuży zarówno dyżurnemu ruchu PKS, jak i pasażerom.
3. Nauka pracy w zespole projektowym.
4. Rozwiązanie zespołu po pomyślnym zaliczeniu projektu. Przewidywany termin to 7 czerwca 2016 r.

Obowiązki członków zespołu:

1. Damian Berda (grafik).
2. Jakub Wachowiak (analityk, projektant).
3. Łukasz Stolarczyk (dokumentalista).
4. Marcin Chruścicki (kierownik, programista).
5. Michał Motykowski (tester).

2. Instrumentarium zespołu projektowego.

W ramach pracy zespołowej zostało stworzone repozytorium projektu o nazwie „DamberPol” na serwerach GitHub.com i Trello.com. Do repozytorium dodawane będą kolejne wersje aplikacji oraz dokumentacji, z możliwością przeglądania przez wszystkich członków zespołu projektowego.

Żeby wspomóc i usprawnić wspólną pracę nad projektem, członkowie zespołu będą kontaktować się za pomocą takich aplikacji, jak Skype i Facebook Messenger.

Narzędzia wykorzystywane podczas pracy projektowej:

- ArgoUML – producent University of California, licencja bezpłatna. (http://argouml.tigris.org)
- Eclipse – producent Eclipse Foundation, licencja Eclipse Public License. (http://www.eclipse.org)
- Microsoft Office Excel – producent Microsoft, licencja komercyjna. (http://www.office.microsoft.com)
- Microsoft Office Word – producent Microsoft, licencja komercyjna. (http://www.office.microsoft.com)
- Microsoft SQL Server – producent Microsoft, licencja Microsoft EULA. (http://www.microsoft.com/sql)
- Microsoft SQL Server Management Studio Express – producent Microsoft, licencja Microsoft EULA. (https://www.microsoft.com/en-us/server-cloud/products/sql-server)
- Mozilla Firefox – producent Mozilla Corporation, licencja Mozilla Licensing. (http://www.mozilla.org)

3. Opis wymagań klienta.

3.1. Model klienta.

Klientem jest Państwowa Komunikacja Samochodowa w Koszalinie. Działalność tej instytucji polega na gromadzeniu pewnego rodzaju danych i udostępnianiu jej części pasażerom za pośrednictwem lokalnej aplikacji.

3.2. Wymaganie funkcjonalne.

- Sprawdzanie połączeń przez pasażerów.
- Sporządzanie raportów (np. tygodniowych, miesięcznych) na temat ilości pasażerów korzystających z danego kursu.
- Zarządzanie (np. dodawanie, usuwanie) listą zatrudnionych kierowców i wykorzystywanych autokarów.
- Administrowanie danymi – wprowadzanie / usuwanie podstawowych obiektów bazy danych.

3.3. Wymagania niefunkcjonalne (ograniczenia).

- Dla każdego kierunku (np. Koszalin – Gdańsk) istnieją kursy wyznaczające realizację danego połączenia (np. data, czas przyjazdu / odjazdu).
- Na danej trasie (np. Koszalin – Gdańsk) istnieją kursy pośrednie (np. Słupsk – Gdańsk).
- Pasażerowie mają możliwość sprawdzenia połączenia poprzez podanie miejscowości początkowej i końcowej oraz dnia, w którym zamierzają podróżować.
- Dyżurny powinien mieć możliwość sporządzenia odpowiednich raportów.

4. Wstępny harmonogram prac.

luty:
- Założenie grupy projektowej.
- Uzgodnienie wstępnych założeń projektu.

marzec:
- Określenie wymagań funkcjonalnych projektu.
- Tworzenie dokumentacji (diagram klas, przypadków użycia, funkcjonalności).

kwiecień:
- Zaprojektowanie graficznego interfejsu użytkownika.
- Programowanie kolejnych etapów projektu.
- Konsultacje z klientem (prowadzącym zajęcia).

maj:
- Dokończenie prac nad projektem.
- Testowanie i naprawa ewentualnych błędów w projekcie.
- Finalizacja prac związanych z projektem.

czerwiec:
- Zaprezentowanie aplikacji.
- Zakończenie prac nad projektem.

5. Specyfikacja wymagań.

5.1. Wstęp.

Celem dokumentu jest specyfikacja obsługi bazodanowej aplikacji w firmie transportowej.

5.1.1. Cele.

- Zaliczenie przedmiotu „projekt zespołowy CASE”.
- Wykonanie projektu aplikacji wspomagającej funkcjonowanie Państwowej Komunikacji Samochodowej.
- Nauka pracy w zespole projektowym.
- Rozwiązanie zespołu po pomyślnym zakończeniu projektu.
- Usprawnienie działania firmy.

5.1.2. Definicje, skróty. Opis.

- Dodawanie kierowców / autokarów – dodawanie zatrudnionych kierowców i wykorzystywanych autokarów w firmie.
- Dyżurny ruchu – wykwalifikowany pracownik, którego obowiązkiem jest prowadzenie ruchu.
- Pasażer – osoba podróżująca pojazdem.

- PKS – Państwowa Komunikacja Samochodowa, firma, dla której opracowywany jest projekt.
- Połączenie – łączność między miastami utrzymywana za pomocą środka lokomocji.
- Sporządzanie raportów – jaki kierowca jest w trasie, jakim jedzie kursem, jaki autokar prowadzi.
- Usuwanie kierowców / autokarów – usuwanie zwolnionych kierowców i niewykorzystywanych autokarów w firmie.

5.2. Ogólny opis.

5.2.1. Przydatność projektowanej aplikacji. Walory użytkowe.

Aplikacja będzie zaprojektowana w taki sposób, aby użytkownik z podstawowymi umiejętnościami obsługi komputera, nie miał jakichkolwiek problemów z obsługą rozkładu jazdy, a dyżurny z nadzorem ruchu.

5.2.2. Możliwości projektowanego oprogramowania.

Aplikacja będzie oferowała:

- Dodawanie autokarów.
- Dodawanie kierowców.
- Dodawanie cyklicznych raportów nt. ilości pasażerów korzystających z danego kursu.
- Sprawdzanie rozkładu jazdy przez pasażerów.
- Usuwanie autokarów.
- Usuwanie kierowców.

5.2.3. Ogólne ograniczenia.

Brak ograniczeń.

5.2.4. Charakterystyka użytkowników.

- Dyżurny ruchu – osoba zarządzająca aplikacją, posiada największe uprawnienia.
- Pasażer – osoba posiadająca ograniczone przywileje, posiada możliwość sprawdzenia rozkładu jazdy autobusów.

5.2.5. Środowisko operacyjne.

Każde środowisko obsługujące wirtualny silnik Java.

5.3. Specyfika wymagań.

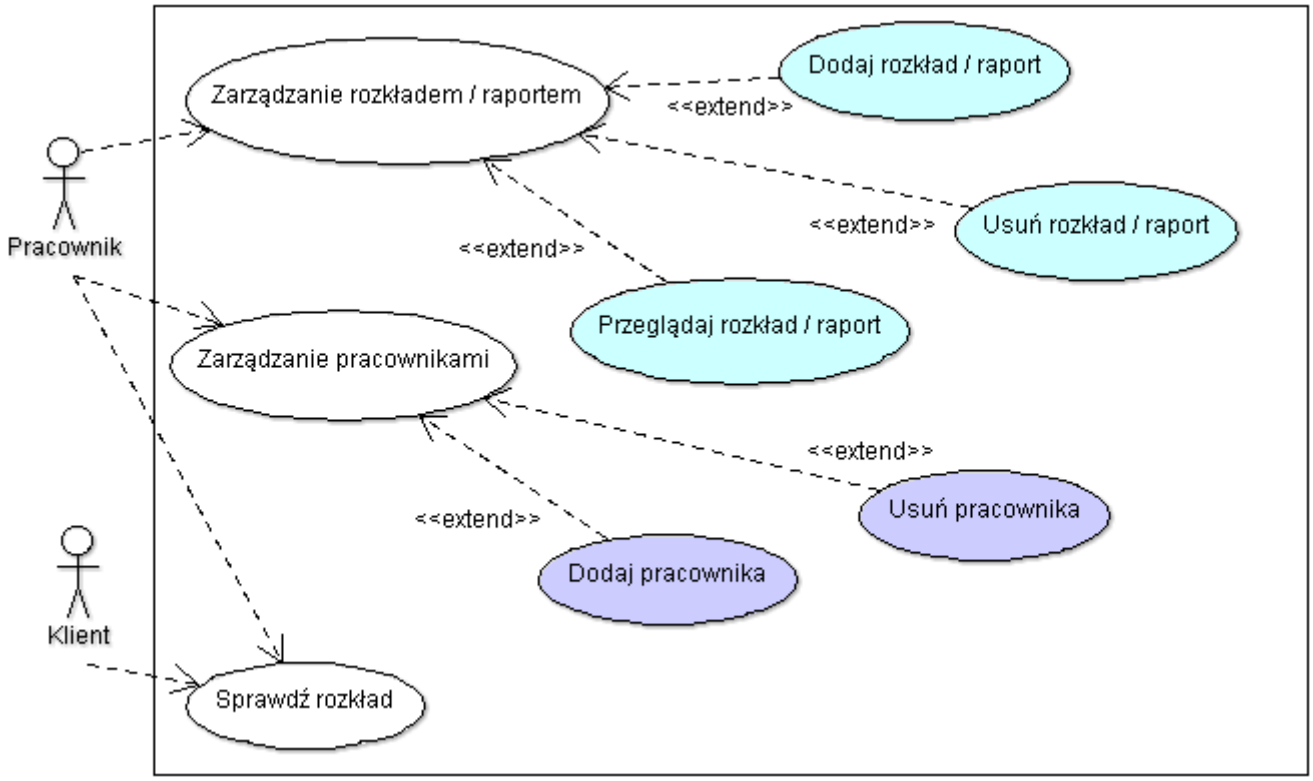
5.3.1. Wymagania funkcjonalne.

- Dodawanie i usuwanie autokarów z bazy danych.
- Dodawanie i usuwanie kierowców z bazy danych.
- Dodawanie i usuwanie z bazy danych informacji o kursie.
- Logowanie do bazy.
- Wyświetlanie rozkładu jazdy.

5.3.2. Wymagania niefunkcjonalne (ograniczenia).

- Pasażer nie ma uprawnień do modyfikacji rozkładu.
- Użytkownik korzysta ze środowiska obsługującego wirtualny silnik Java.

5.4. Diagram przypadków użycia.



6. Raport dla zleceniodawcy.

6.1. Cele przedsięwzięcia.

Celem projektu jest stworzenie aplikacji wspomagającej działanie firmy transportowej. Program ma na celu ułatwić pracownikom zarządzanie danymi, a czytelnikom zapewnić możliwość przeglądania wybranych zasobów, rozkładu jazdy.

6.2. Zakres przedsięwzięcia.

- Określenie wymagań ze strony klienta.
- Analiza i projektowanie systemu.
- Realizacja projektu.
- Tworzenie dokumentacji.
- Testowanie.
- Wdrożenie aplikacji.

6.3. Systemy zewnętrzne.

Aplikacja będzie zgodna z systemem operacyjnym Windows (XP, Vista, Windows 7, Windows 8, Windows 10).

Oprogramowanie będzie współpracować z systemem bazodanowym MS SQL.

6.4. Szkicowy opis wymagań.

6.4.1. Wymagania funkcjonalne.

- Zarządzanie rozkładem jazdy.
- Zarządzanie kontami użytkowników.
- Zdalne przeglądanie zasobów rozkładu jazdy.

6.4.2. Wymagania niefunkcjonalne (ograniczenia).

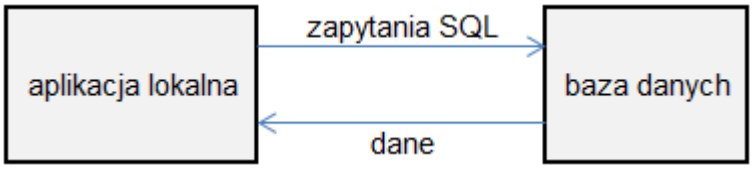
- Weryfikacja użytkownika.

6.5. Szkicowy opis rozwiązań.

Realizacja aplikacji lokalnej zostanie wykonana w technologii Java. Do przygotowania bazy danych wykorzystany zostanie Microsoft SQL Server oraz Microsoft SQL Server Management Studio.

6.6. Szkicowy opis modelu systemu.

System będzie się składał z aplikacji lokalnej. Aplikacja będzie się łączyć z bazą danych Microsoft Server.



6.7. Kosztorys projektu.

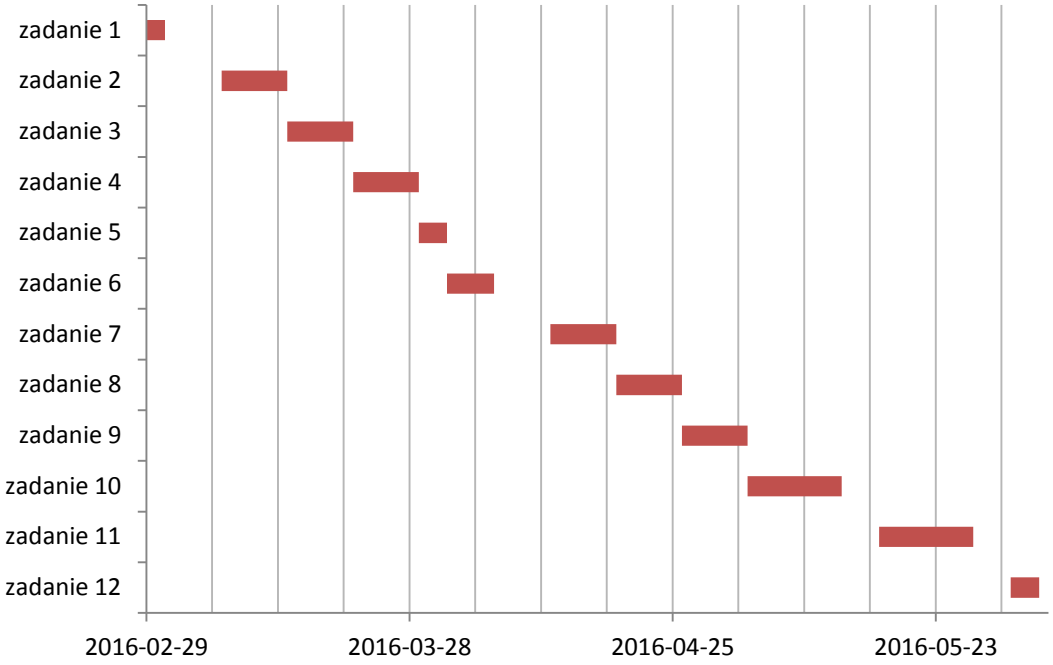
koszty zakupu oprogramowania							
lp.	nazwa	wersja	licencja	cena	ilość	czas realizacji	suma
1	Microsoft Windows	7	komercyjna	599 zł	5	4 mies.	2 995 zł
2	Microsoft Office	365 BP	komercyjna	45 zł / mies.	5		900 zł
3	Eclipse	Java EE Dev.	GPL	-	4		-
4	XAMPP	7.0.4	GPL	-	5		-
5	SAP S. PowerDesigner	16.5	komercyjna	10 467 zł	1		10 467 zł
6	GitHub	Silver	komercyjna	190 zł / mies.	1		760 zł
7	Git Large File Storage	-	komercyjna	19 zł / mies.	1		76 zł
łączny koszt zakupu oprogramowania							**15 198 zł**

koszty pracy				
lp.	zadanie	cena za godz.	czas realizacji	suma
1	analiza wymagań	54 zł	91 godz.	4 914 zł
2	projekt systemu	62 zł	150 godz.	9 300 zł
3	realizacja aplikacji	70 zł	165 godz.	11 550 zł
4	testowanie aplikacji	60 zł	85 godz.	5 100 zł
5	tworzenie dokumentacji	45 zł	90 godz.	4 050 zł
łączny koszt pracy				**34 914 zł**

koszty utrzymania				
lp.	usługa	miesięczny koszt	czas realizacji	suma
1	wynajem lokalu	600 zł	4 mies.	2 400 zł
2	energia elektryczna	400 zł		1 600 zł
3	dostęp do internetu	80 zł		320 zł
4	ogrzewanie	300 zł		1 200 zł
5	dojazdy	600 zł		2 400 zł
6	licencja ZAiKS	81 zł		324 zł
7	komputer (7x)	0 zł (środki własne)		0 zł
8	pracownicy : 2x młodszy programista 2x starszy programista 1x kierownik projektu	12 000 zł / 2x os. 20 000 zł / 2x os. 10 000 zł / os.		168 000 zł
łączny koszt utrzymania				**176 244 zł**

łączny koszt realizacji projektu	**226 356 zł**

6.8. Wstępny harmonogram prac.



Zadanie 1 (29.02 – 01.03). Uzgodnienie tematu projektu, wstępna umowa z klientem. Utworzenie grupy projektowej, ustalanie zasad współpracy z członkami grupy. Utworzenie protokołu z zebrania założycielskiego.

Zadanie 2 (08.03 – 15.03). Utworzenie harmonogramu prac. Opracowanie dokumentacji wymagań użytkownika. Raport klienta, opracowanie raportu dla zleceniodawcy (zakres przedsięwzięcia; przygotowanie szkicowego opisu wymagań, modelu systemu, proponowanych rozwiązań, oszacowanie kosztów projektu oraz daty zakończenia prac nad projektem).

Zadanie 3 (15.03 – 22.03). Określenie wymagań funkcjonalnych projektu, opracowanie specyfikacji wymagań (opracowanie słownika pojęć, przygotowanie listy wymagań funkcjonalnych i niefunkcjonalnych, zidentyfikowanie aktorów oraz ich funkcjonalności, przeprowadzenie identyfikacji przypadków użycia, opracowanie diagramu przypadków użycia). Opracowanie stylu firmowego projektu (logo, wizytówka). Zaprojektowanie modelu bazy danych (model konceptualny i model fizyczny), stworzenie bazy danych na potrzeby projektu.

Zadanie 4 (22.03 – 29.03). Wybór odpowiednich narzędzi CASE. Opracowanie zestawu diagramów (przypadków użycia, klas, przepływu danych).

Zadanie 5 (29.03 – 01.04). Wykończenie prac nad stylem firmowym projektu (logo, wizytówka).

Zadanie 6 (01.04 – 06.04). Zaprojektowanie graficznego interfejsu użytkownika. Opracowanie słownika danych zawierający specyfikację modelu.

Zadanie 7 (12.04 – 19.04). Opracowanie diagramu związków encji, diagramu przejść stanów, diagramu aktywności, diagramu sekwencji, diagramu współpracy, diagramu komponentów. Analiza kosztorysu. Konsultacje z klientem.

Zadanie 8 (19.04 – 26.04). Wybór odpowiednich narzędzi do szybkiego rozwijania aplikacji (ang. RAD). Ciąg dalszy prac nad projektem (zaprogramowanie GUI, podstawowych funkcjonalności).

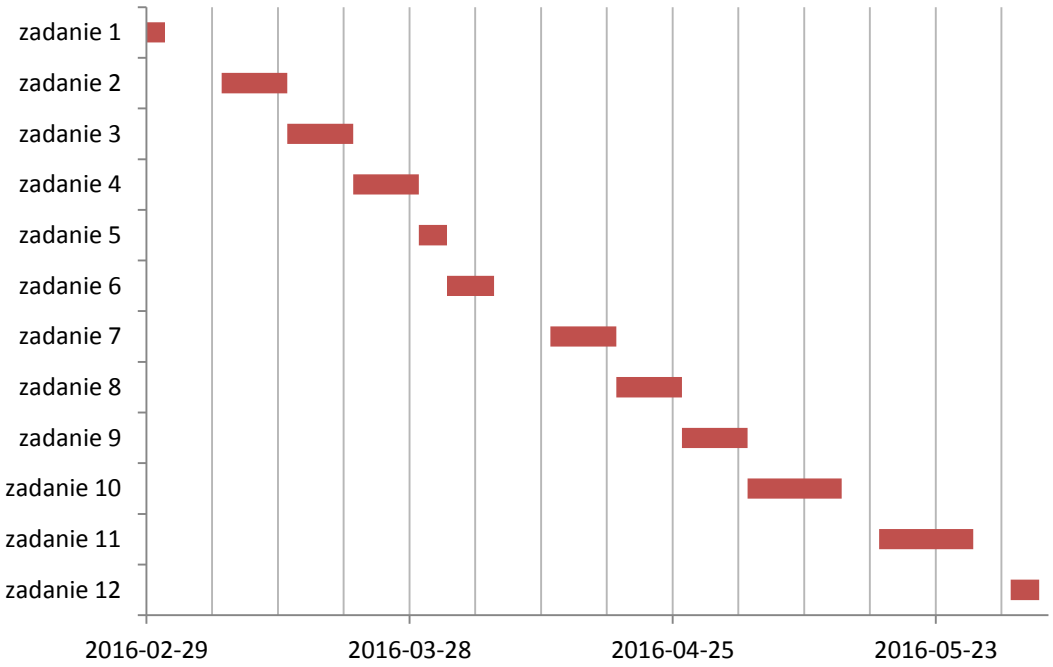
Zadanie 9 (26.04 – 03.05). Prace nad programowaniem dalszych funkcjonalności systemowych.

Zadanie 10 (03.05 – 13.05). Wstępne testowanie aplikacji. Naprawa ewentualnych błędów.

Zadanie 11 (17.05 – 27.05). Zakończenie programowania projektu. Przeprowadzenie szczegółowych testów. Finalizacja projektu. Sprawdzenie i zamknięcie dokumentacji. Przygotowanie projektu do zaprezentowania klientowi.

Zadanie 12 (31.05 – 03.06). Prezentacja projektu klientowi. Zakończenie prac nad projektem.

7. Harmonogram prac.



Zadanie 1 (29.02 – 01.03). Uzgodnienie tematu projektu, wstępna umowa z klientem. Utworzenie grupy projektowej, ustalanie zasad współpracy z członkami grupy. Utworzenie protokołu z zebrania założycielskiego.

Zadanie 2 (08.03 – 15.03). Utworzenie harmonogramu prac. Opracowanie dokumentacji wymagań użytkownika. Raport klienta, opracowanie raportu dla zleceniodawcy (zakres przedsięwzięcia; przygotowanie szkicowego opisu wymagań, modelu systemu, proponowanych rozwiązań, oszacowanie kosztów projektu oraz daty zakończenia prac nad projektem).

Zadanie 3 (15.03 – 22.03). Określenie wymagań funkcjonalnych projektu, opracowanie specyfikacji wymagań (opracowanie słownika pojęć, przygotowanie listy wymagań funkcjonalnych

i niefunkcjonalnych, zidentyfikowanie aktorów oraz ich funkcjonalności, przeprowadzenie identyfikacji przypadków użycia, opracowanie diagramu przypadków użycia). Opracowanie stylu firmowego projektu (logo, wizytówka). Zaprojektowanie modelu bazy danych (model konceptualny i model fizyczny), stworzenie bazy danych na potrzeby projektu.

Zadanie 4 (22.03 – 29.03). Wybór odpowiednich narzędzi CASE. Opracowanie zestawu diagramów (przypadków użycia, klas, przepływu danych).

Zadanie 5 (29.03 – 01.04). Wykończenie prac nad stylem firmowym projektu (logo, wizytówka).

Zadanie 6 (01.04 – 06.04). Zaprojektowanie graficznego interfejsu użytkownika. Opracowanie słownika danych zawierający specyfikację modelu.

Zadanie 7 (12.04 – 19.04). Opracowanie diagramu związków encji, diagramu przejść stanów, diagramu aktywności, diagramu sekwencji, diagramu współpracy, diagramu komponentów. Analiza kosztorysu. Konsultacje z klientem.

Zadanie 8 (19.04 – 26.04). Wybór odpowiednich narzędzi do szybkiego rozwijania aplikacji (ang. RAD). Ciąg dalszy prac nad projektem (zaprogramowanie GUI, podstawowych funkcjonalności).

Zadanie 9 (26.04 – 03.05). Prace nad programowaniem dalszych funkcjonalności systemowych.

Zadanie 10 (03.05 – 13.05). Wstępne testowanie aplikacji. Naprawa ewentualnych błędów.

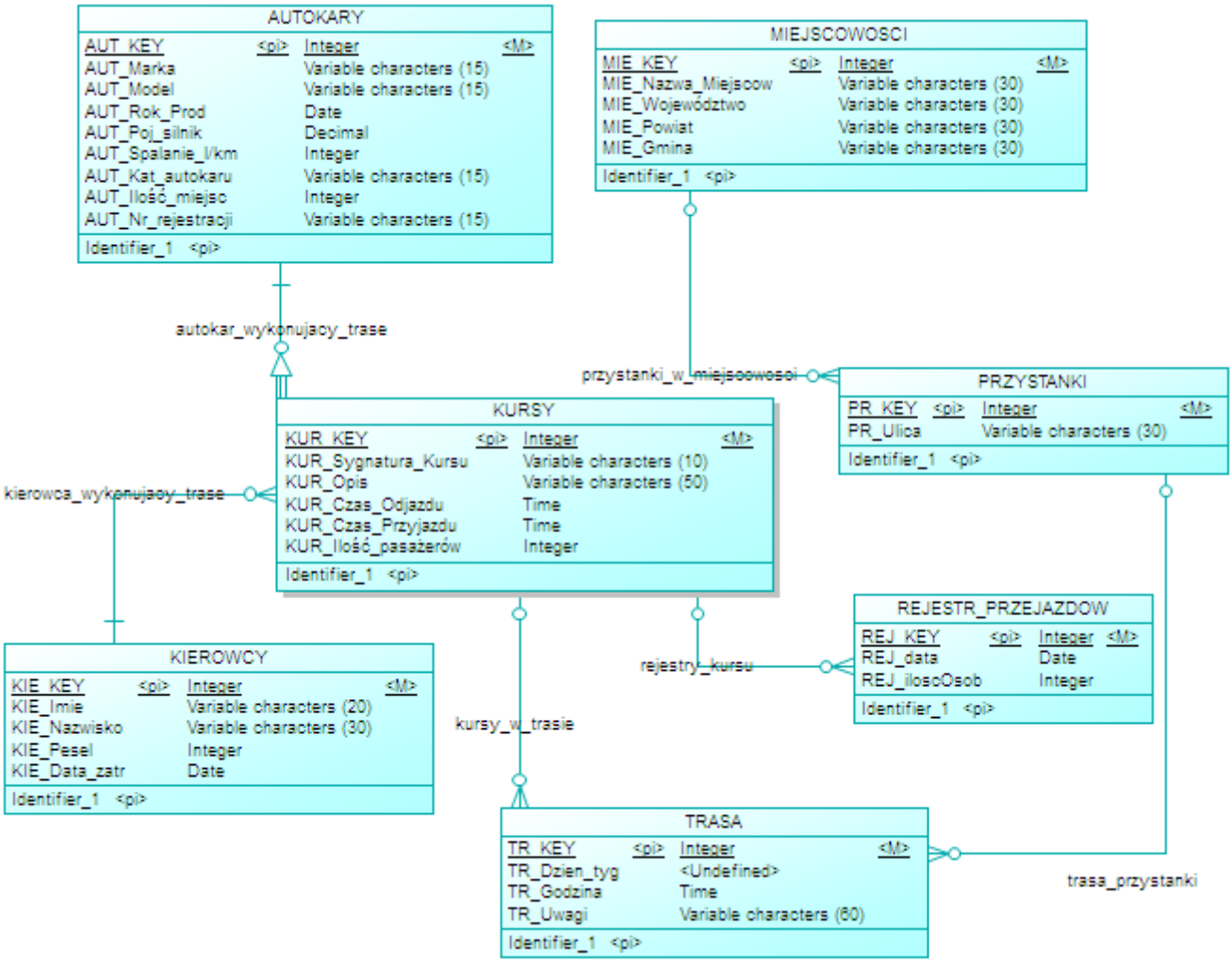
Zadanie 11 (17.05 – 27.05). Zakończenie programowania projektu. Przeprowadzenie szczegółowych testów. Finalizacja projektu. Sprawdzenie i zamknięcie dokumentacji. Przygotowanie projektu do zaprezentowania klientowi.

Zadanie 12 (31.05 – 03.06). Prezentacja projektu klientowi. Zakończenie prac nad projektem.

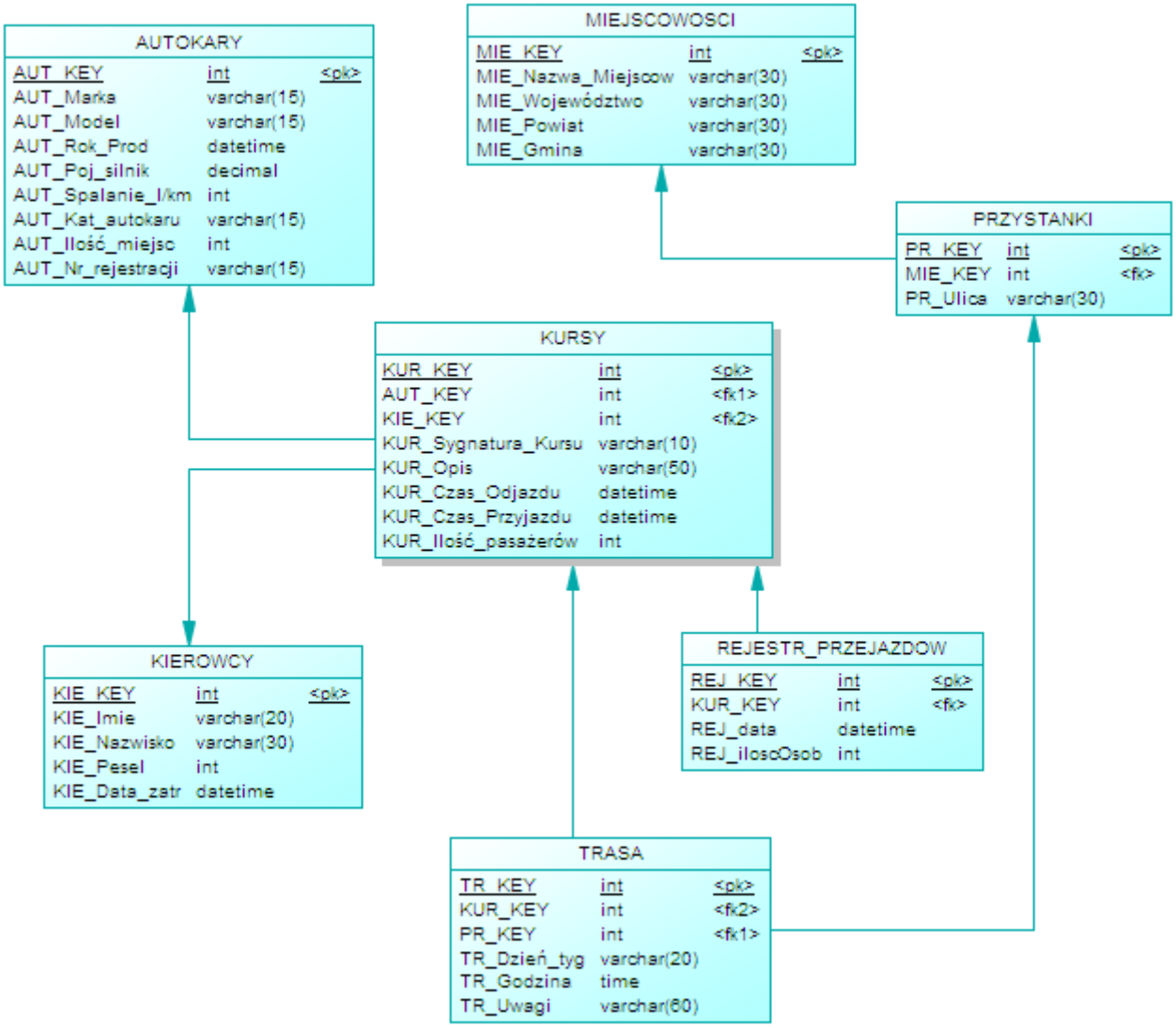
8. Wizytówka.

9. Logo.

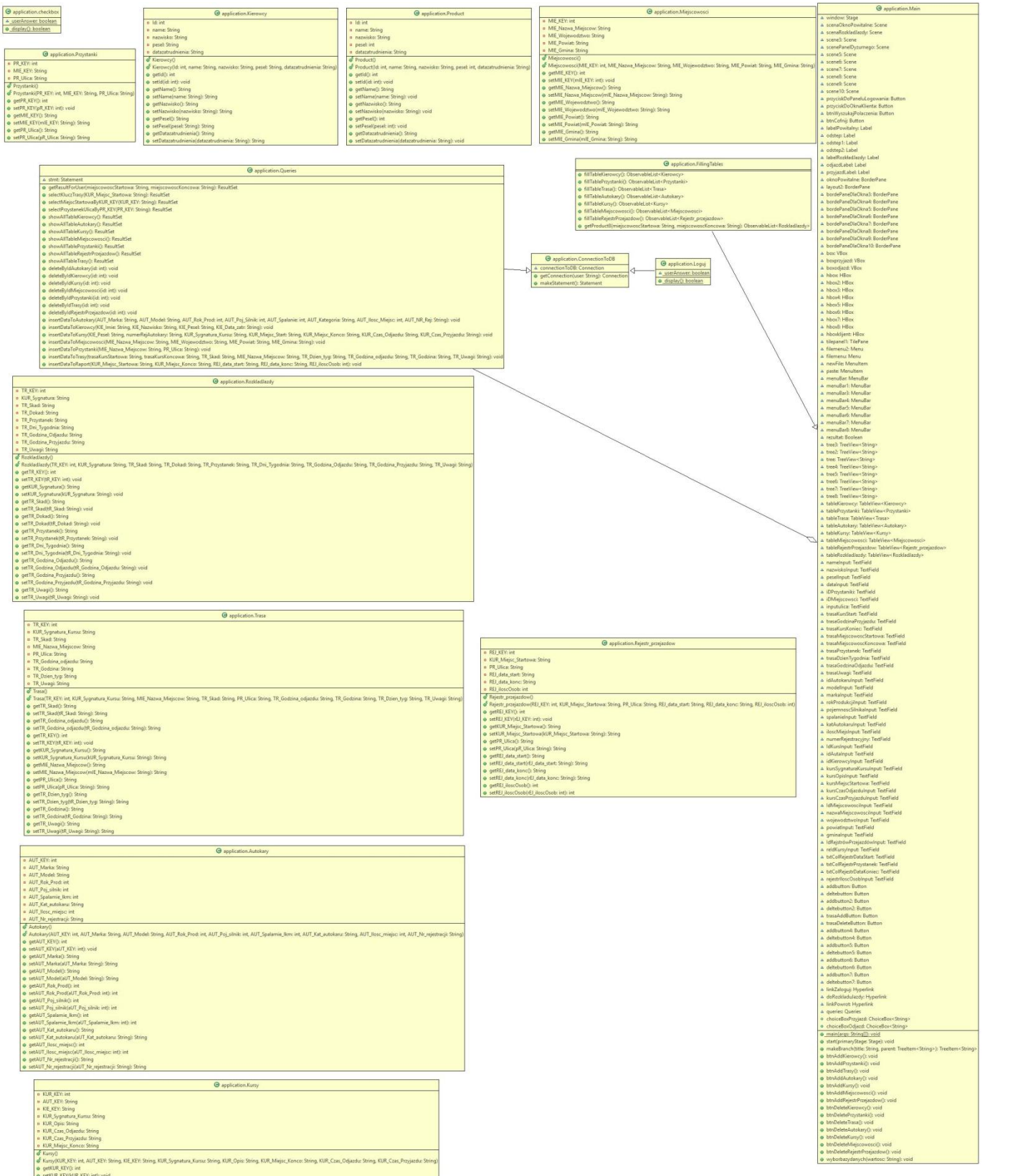
10. Model konceptualny bazy danych.



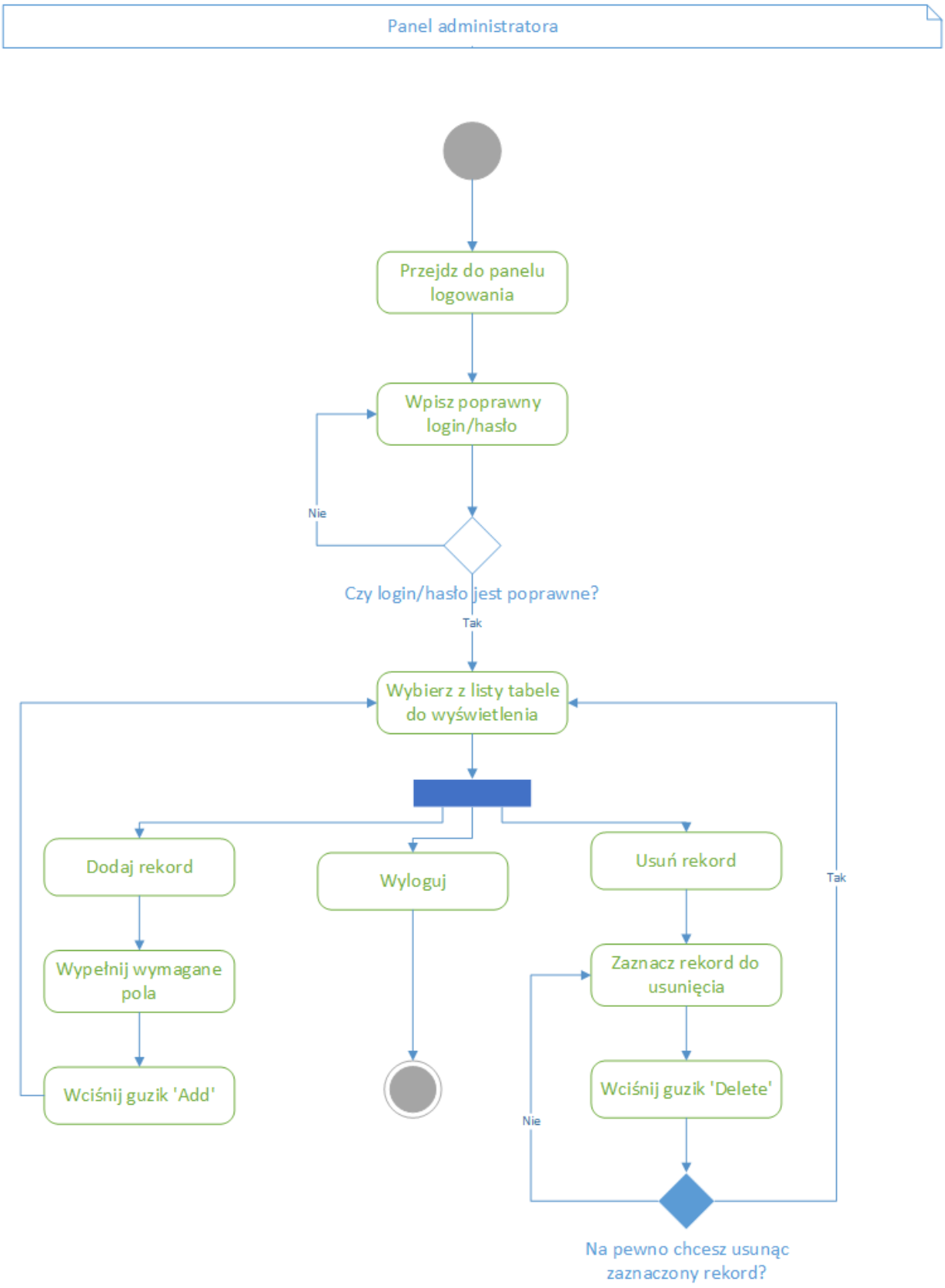
11. Model fizyczny bazy danych.

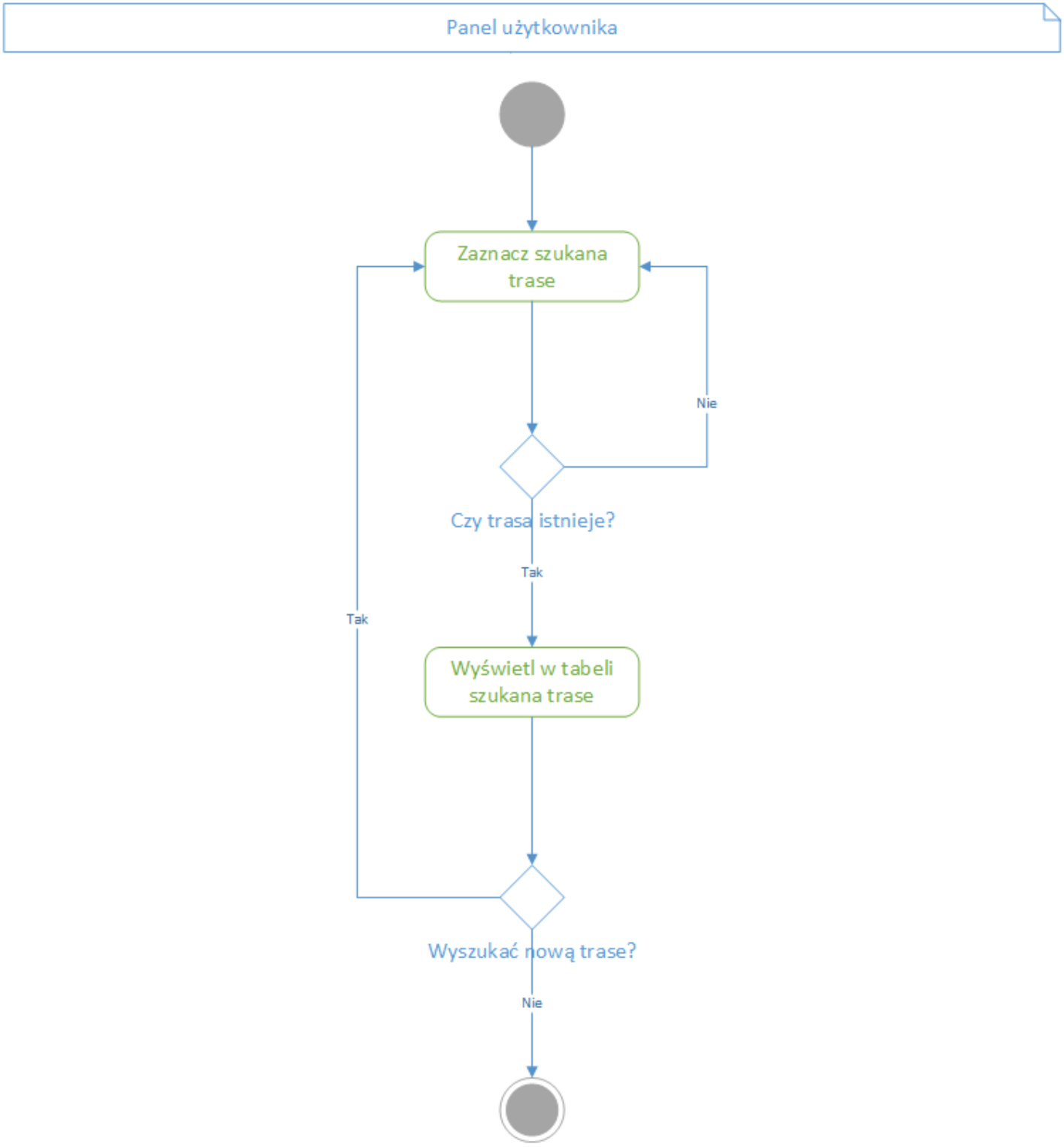


12. Diagram klas.

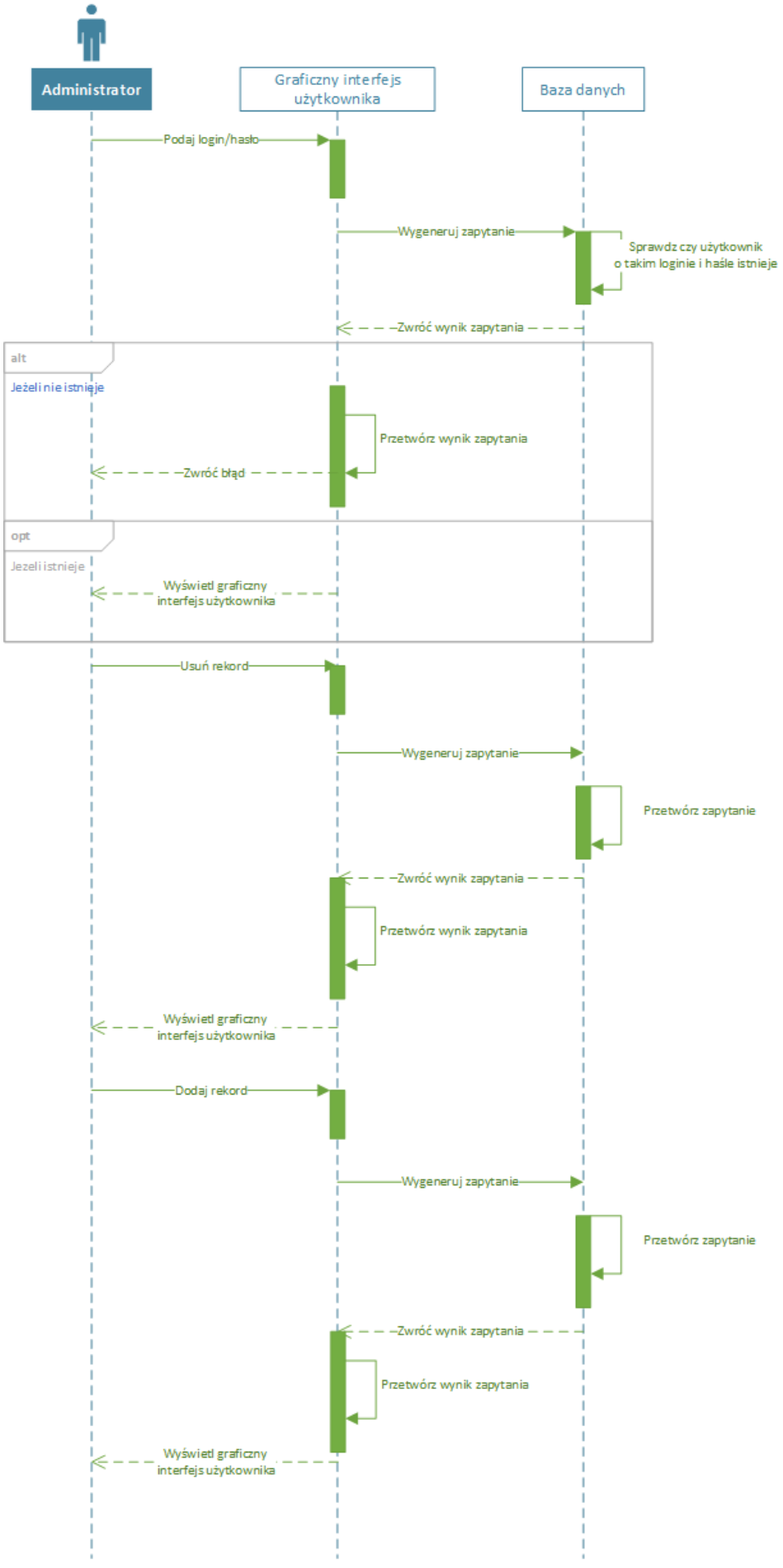

application.checkbox
userAnswer: boolean
display(): boolean
application.Przystanki
PR_KEY: int
MIE_KEY: String
PR_Ulica: String
Przystanki()
Przystanki(PR_KEY: int, MIE_KEY: String, PR_Ulica: String)
getPR_KEY(): int
setPR_KEY(pR_KEY: int): void
getMIE_KEY(): String
setMIE_KEY(mIE_KEY: String): String
getPR_Ulica(): String
setPR_Ulica(pR_Ulica: String): String
application.Kierowcy
Id: int
name: String
nazwisko: String
pesel: String
datazatrudnienia: String
Kierowcy()
Kierowcy(Id: int, name: String, nazwisko: String, pesel: String, datazatrudnienia: String)
getId(): int
setId(id: int): void
getName(): String
setName(name: String): String
getNazwisko(): String
setNazwisko(nazwisko: String): String
getPesel(): String
setPesel(pesel: String): String
getDatazatrudnienia(): String
setDatazatrudnienia(datazatrudnienia: String): String
application.Product
Id: int
name: String
nazwisko: String
pesel: int
datazatrudnienia: String
Product()
Product(Id: int, name: String, nazwisko: String, pesel: int, datazatrudnienia: String)
getId(): int
setId(id: int): void
getName(): String
setName(name: String): void
getNazwisko(): String
setNazwisko(nazwisko: String): void
getPesel(): int
setPesel(pesel: int): void
getDatazatrudnienia(): String
setDatazatrudnienia(datazatrudnienia: String): void
application.Miejscowosci
MIE_KEY: int
MIE_Nazwa_Miejscow: String
MIE_Wojewodztwo: String
MIE_Powiat: String
MIE_Gmina: String
Miejscowosci()
Miejscowosci(MIE_KEY: int, MIE_Nazwa_Miejscow: String, MIE_Wojewodztwo: String, MIE_Powiat: String, MIE_Gmina: String)
getMIE_KEY(): int
setMIE_KEY(mIE_KEY: int): void
getMIE_Nazwa_Miejscow(): String
setMIE_Nazwa_Miejscow(mIE_Nazwa_Miejscow: String): String
getMIE_Wojewodztwo(): String
setMIE_Wojewodztwo(mIE_Wojewodztwo: String): String
getMIE_Powiat(): String
setMIE_Powiat(mIE_Powiat: String): String
getMIE_Gmina(): String
setMIE_Gmina(mIE_Gmina: String): String
application.Main
window: Stage
scenaOknoPowitalne: Scene
scenaRozkladJazdy: Scene
scene3: Scene
scenePanelDyzurnego: Scene
scene5: Scene
scene6: Scene
scene7: Scene
scene8: Scene
scene9: Scene
scene10: Scene
przyciskDoPaneluLogowania: Button
przyciskDoOknaKlienta: Button
btnWyszukajPolaczenia: Button
btnCofnij: Button
labelPowitalny: Label
odstep: Label
odstep1: Label
odstep2: Label
labelRozkladJazdy: Label
odjazdLabel: Label
przyjazdLabel: Label
oknoPowitalne: BorderPane
layout2: BorderPane
bordePaneDlaOkna3: BorderPane
bordePaneDlaOkna4: BorderPane
bordePaneDlaOkna5: BorderPane
bordePaneDlaOkna6: BorderPane
bordePaneDlaOkna7: BorderPane
bordePaneDlaOkna8: BorderPane
bordePaneDlaOkna9: BorderPane
bordePaneDlaOkna10: BorderPane
box: VBox
boxprzyjazd: VBox
boxodjazd: VBox
hbox: HBox
hbox2: HBox
hbox3: HBox
hbox4: HBox
hbox5: HBox
hbox6: HBox
hbox7: HBox
hbox8: HBox
hboxKlient: HBox
tilepane11: TilePane
filemenu2: Menu
filemenu: Menu
newFile: MenuItem
paste: MenuItem
menuBar: MenuBar
menuBar1: MenuBar
menuBar3: MenuBar
menuBar4: MenuBar
menuBar5: MenuBar
menuBar6: MenuBar
menuBar7: MenuBar
menuBar8: MenuBar
rezultat: Boolean
tree3: TreeView<String>
tree2: TreeView<String>
tree: TreeView<String>
tree4: TreeView<String>
tree5: TreeView<String>
tree6: TreeView<String>
tree7: TreeView<String>
tree8: TreeView<String>
tableKierowcy: TableView<Kierowcy>
tablePrzystanki: TableView<Przystanki>
tableTrasa: TableView<Trasa>
tableAutokary: TableView<Autokary>
tableKursy: TableView<Kursy>
tableMiejscowosci: TableView<Miejscowosci>
tableRejestrPrzejazdow: TableView<Rejestr_przejazdow>
tableRozkladJazdy: TableView<RozkladJazdy>
nameInput: TextField
nazwiskoInput: TextField
peselInput: TextField
dataInput: TextField
iDPrzystanki: TextField
iDMiejscowosci: TextField
inputulica: TextField
trasaKursStart: TextField
trasaGodzinaPrzyjazdu: TextField
trasaKursKoniec: TextField
trasaMiejscowoscStartowa: TextField
trasaMiejscowoscKoncowa: TextField
trasaPrzystanek: TextField
trasaDzienTygodnia: TextField
trasaGodzinaOdjazdu: TextField
trasaUwagi: TextField
idAutokaruInput: TextField
modelInput: TextField
markaInput: TextField
rokProdukcjiInput: TextField
pojemnoscSilnikaInput: TextField
spalanieInput: TextField
katAutokaruInput: TextField
iloscMiejscInput: TextField
numerRejestracyjny: TextField
idKursInput: TextField
idKatalnput: TextField
idKierowcyInput: TextField
kursSygnaturaKursuInput: TextField
kursOpisInput: TextField
kursMiejscStartowa: TextField
kursCzasOdjazduInput: TextField
kursCzasPrzyjazduInput: TextField
idMiejscowosciInput: TextField
nazwaMiejscowosciInput: TextField
wojewodztwoInput: TextField
powiatInput: TextField
gminaInput: TextField
idRejestrowPrzejazdowInput: TextField
reldKursyInput: TextField
btColRejestrDataStart: TextField
btColRejestrPrzystanek: TextField
btColRejestrDataKoniec: TextField
rejestrIloscOsobInput: TextField
addbutton: Button
deltebutton: Button
addbutton2: Button
deltebutton2: Button
trasaAddButton: Button
trasaDeleteButton: Button
addbutton4: Button
deltebutton4: Button
addbutton5: Button
deltebutton5: Button
addbutton6: Button
deltebutton6: Button
addbutton7: Button
deltebutton7: Button
linkZaloguj: Hyperlink
doRozkladuJazdy: Hyperlink
linkPowrot: Hyperlink
queries: Queries
choiceBoxPrzyjazd: ChoiceBox<String>
choiceBoxOdjazd: ChoiceBox<String>
main(args: String[]): void
start(primaryStage: Stage): void
makeBranch(title: String, parent: TreeItem<String>): TreeItem<String>
btnAddKierowcy(): void
btnAddPrzystanki(): void
btnAddTrasy(): void
btnAddAutokary(): void
btnAddKursy(): void
btnAddMiejscowosci(): void
btnAddRejestrPrzejazdow(): void
btnDeleteKierowcy(): void
btnDeletePrzystanki(): void
btnDeleteTrasa(): void
btnDeleteAutokary(): void
btnDeleteKursy(): void
btnDeleteMiejscowosci(): void
btnDeleteRejestrPrzejazdow(): void
wyborbazydanych(wartosc: String): void
application.Queries
stmt: Statement
getResultForUser(miejscowoscStartowa: String, miejscowoscKoncowa: String): ResultSet
selectKtoczTrasy(KUR_Miejsc_Startowa: String): ResultSet
selectMiejscStartowaByKUR_KEY(KUR_KEY: String): ResultSet
selectPrzystanekUlicaByPR_KEY(PR_KEY: String): ResultSet
showAllTableKierowcy(): ResultSet
showAllTableAutokary(): ResultSet
showAllTableKursy(): ResultSet
showAllTableMiejscowosci(): ResultSet
showAllTablePrzystanki(): ResultSet
showAllTableRejestrPrzejazdow(): ResultSet
showAllTableTrasy(): ResultSet
deleteByIdAutokary(id: int): void
deleteByIdKierowcy(id: int): void
deleteByIdKursy(id: int): void
deleteByIdMiejscowosci(id: int): void
deleteByIdPrzystanki(id: int): void
deleteByIdTrasy(id: int): void
deleteByIdRejestrPrzejazdow(id: int): void
insertDataToAutokary(AUT_Marka: String, AUT_Model: String, AUT_Rok_Prod: int, AUT_Poj_Silnik: int, AUT_Spalanie: int, AUT_Kategoria: String, AUT_Ilosc_Miejsc: int, AUT_NR_Rej: String): void
insertDataToKierowcy(KIE_Imie: String, KIE_Nazwisko: String, KIE_Pesel: String, KIE_Data_zatr: String): void
insertDataToKursy(KIE_Pesel: String, numerRejAutokary: String, KUR_Sygnatura_Kursu: String, KUR_Miejsc_Start: String, KUR_Miejsc_Konco: String, KUR_Czas_Odjazdu: String, KUR_Czas_Przyjazdu: String): void
insertDataToMiejscowosci(MIE_Nazwa_Miejscow: String, MIE_Wojewodztwo: String, MIE_Powiat: String, MIE_Gmina: String): void
insertDataToPrzystanki(MIE_Nazwa_Miejscow: String, PR_Ulica: String): void
insertDataToTrasy(trasaKursStartowa: String, TR_Skad: String, MIE_Nazwa_Miejscow: String, TR_Dzien_tyg: String, TR_Godzina_odjazdu: String, TR_Godzina: String, TR_Uwagi: String): void
insertDataToRaport(KUR_Miejsc_Startowa: String, KUR_Miejsc_Konco: String, REJ_data_start: String, REJ_data_konc: String, REJ_IloscOsob: int): void
application.FillingTables
fillTableKierowcy(): ObservableList<Kierowcy>
fillTablePrzystanki(): ObservableList<Przystanki>
fillTableTrasa(): ObservableList<Trasa>
fillTableAutokary(): ObservableList<Autokary>
fillTableKursy(): ObservableList<Kursy>
fillTableMiejscowosci(): ObservableList<Miejscowosci>
fillTableRejestrPrzejazdow(): ObservableList<Rejestr_przejazdow>
getProduct8(miejscowoscStartowa: String, miejscowoscKoncowa: String): ObservableList<RozkladJazdy>
application.ConnectionToDB
connectionToDB: Connection
getConnection(user: String): Connection
makeStatement(): Statement
application.Loguj
userAnswer: boolean
display(): boolean
application.RozkladJazdy
TR_KEY: int
KUR_Sygnatura: String
TR_Skad: String
TR_Dokad: String
TR_Przystanek: String
TR_Dni_Tygodnia: String
TR_Godzina_Odjazdu: String
TR_Godzina_Przyjazdu: String
TR_Uwagi: String
RozkladJazdy()
RozkladJazdy(TR_KEY: int, KUR_Sygnatura: String, TR_Skad: String, TR_Dokad: String, TR_Przystanek: String, TR_Dni_Tygodnia: String, TR_Godzina_Odjazdu: String, TR_Godzina_Przyjazdu: String, TR_Uwagi: String)
getTR_KEY(): int
setTR_KEY(tR_KEY: int): void
getKUR_Sygnatura(): String
setKUR_Sygnatura(KUR_Sygnatura: String): void
getTR_Skad(): String
setTR_Skad(tR_Skad: String): void
getTR_Dokad(): String
setTR_Dokad(tR_Dokad: String): void
getTR_Przystanek(): String
setTR_Przystanek(tR_Przystanek: String): void
getTR_Dni_Tygodnia(): String
setTR_Dni_Tygodnia(tR_Dni_Tygodnia: String): void
getTR_Godzina_Odjazdu(): String
setTR_Godzina_Odjazdu(tR_Godzina_Odjazdu: String): void
getTR_Godzina_Przyjazdu(): String
setTR_Godzina_Przyjazdu(tR_Godzina_Przyjazdu: String): void
getTR_Uwagi(): String
setTR_Uwagi(tR_Uwagi: String): void
application.Trasa
TR_KEY: int
KUR_Sygnatura_Kursu: String
TR_Skad: String
MIE_Nazwa_Miejscow: String
PR_Ulica: String
TR_Godzina_odjazdu: String
TR_Godzina: String
TR_Dzien_tyg: String
TR_Uwagi: String
Trasa()
Trasa(TR_KEY: int, KUR_Sygnatura_Kursu: String, MIE_Nazwa_Miejscow: String, TR_Skad: String, PR_Ulica: String, TR_Godzina_odjazdu: String, TR_Godzina: String, TR_Dzien_tyg: String, TR_Uwagi: String)
getTR_Skad(): String
setTR_Skad(tR_Skad: String): String
getTR_Godzina_odjazdu(): String
setTR_Godzina_odjazdu(tR_Godzina_odjazdu: String): String
getTR_KEY(): int
setTR_KEY(tR_KEY: int): void
getKUR_Sygnatura_Kursu(): String
setKUR_Sygnatura_Kursu(KUR_Sygnatura_Kursu: String): String
getMIE_Nazwa_Miejscow(): String
setMIE_Nazwa_Miejscow(mIE_Nazwa_Miejscow: String): String
getPR_Ulica(): String
setPR_Ulica(pR_Ulica: String): String
getTR_Dzien_tyg(): String
setTR_Dzien_tyg(tR_Dzien_tyg: String): String
getTR_Godzina(): String
setTR_Godzina(tR_Godzina: String): String
getTR_Uwagi(): String
setTR_Uwagi(tR_Uwagi: String): String
application.Rejestr_przejazdow
REJ_KEY: int
KUR_Miejsc_Startowa: String
PR_Ulica: String
REJ_data_start: String
REJ_data_konc: String
REJ_IloscOsob: int
Rejestr_przejazdow()
Rejestr_przejazdow(REJ_KEY: int, KUR_Miejsc_Startowa: String, PR_Ulica: String, REJ_data_start: String, REJ_data_konc: String, REJ_IloscOsob: int)
getREJ_KEY(): int
setREJ_KEY(rEJ_KEY: int): void
getKUR_Miejsc_Startowa(): String
setKUR_Miejsc_Startowa(KUR_Miejsc_Startowa: String): String
getPR_Ulica(): String
setPR_Ulica(pR_Ulica: String): String
getREJ_data_start(): String
setREJ_data_start(rEJ_data_start: String): String
getREJ_data_konc(): String
setREJ_data_konc(rEJ_data_konc: String): String
getREJ_IloscOsob(): int
setREJ_IloscOsob(rEJ_IloscOsob: int): int
application.Autokary
AUT_KEY: int
AUT_Marka: String
AUT_Model: String
AUT_Rok_Prod: int
AUT_Poj_silnik: int
AUT_Spalanie_lkm: int
AUT_Kat_autokaru: String
AUT_Ilosc_miejsc: int
AUT_Nr_rejestracji: String
Autokary()
Autokary(AUT_KEY: int, AUT_Marka: String, AUT_Model: String, AUT_Rok_Prod: int, AUT_Poj_silnik: int, AUT_Spalanie_lkm: int, AUT_Kat_autokaru: String, AUT_Ilosc_miejsc: int, AUT_Nr_rejestracji: String)
getAUT_KEY(): int
setAUT_KEY(aUT_KEY: int): void
getAUT_Marka(): String
setAUT_Marka(aUT_Marka: String): String
getAUT_Model(): String
setAUT_Model(aUT_Model: String): String
getAUT_Rok_Prod(): int
setAUT_Rok_Prod(AUT_Rok_Prod: int): int
getAUT_Poj_silnik(): int
setAUT_Poj_silnik(aUT_Poj_silnik: int): int
getAUT_Spalanie_lkm(): int
setAUT_Spalanie_lkm(aUT_Spalanie_lkm: int): int
getAUT_Kat_autokaru(): String
setAUT_Kat_autokaru(aUT_Kat_autokaru: String): String
getAUT_Ilosc_miejsc(): int
setAUT_Ilosc_miejsc(aUT_Ilosc_miejsc: int): int
getAUT_Nr_rejestracji(): String
setAUT_Nr_rejestracji(aUT_Nr_rejestracji: String): String
application.Kursy
KUR_KEY: int
AUT_KEY: String
KIE_KEY: String
KUR_Sygnatura_Kursu: String
KUR_Opis: String
KUR_Czas_Odjazdu: String
KUR_Czas_Przyjazdu: String
KUR_Miejsc_Konco: String
Kursy()
Kursy(KUR_KEY: int, AUT_KEY: String, KIE_KEY: String, KUR_Sygnatura_Kursu: String, KUR_Opis: String, KUR_Miejsc_Konco: String, KUR_Czas_Odjazdu: String, KUR_Czas_Przyjazdu: String)
getKUR_KEY(): int
setKUR_KEY(KUR_KEY: int): void
getAUT_KEY(): String
setAUT_KEY(AUT_KEY: String): String
getKIE_KEY(): String
setKIE_KEY(kIE_KEY: String): String
getKUR_Sygnatura_Kursu(): String
setKUR_Sygnatura_Kursu(KUR_Sygnatura_Kursu: String): String
getKUR_Opis(): String
setKUR_Opis(KUR_Opis: String): String
setKUR_Miejsc_Konco(KUR_Miejsc_Konco: String): String
getKUR_Miejsc_Konco(): String
getKUR_Czas_Odjazdu(): String
setKUR_Czas_Odjazdu(KUR_Czas_Odjazdu: String): String
getKUR_Czas_Przyjazdu(): String
setKUR_Czas_Przyjazdu(KUR_Czas_Przyjazdu: String): String

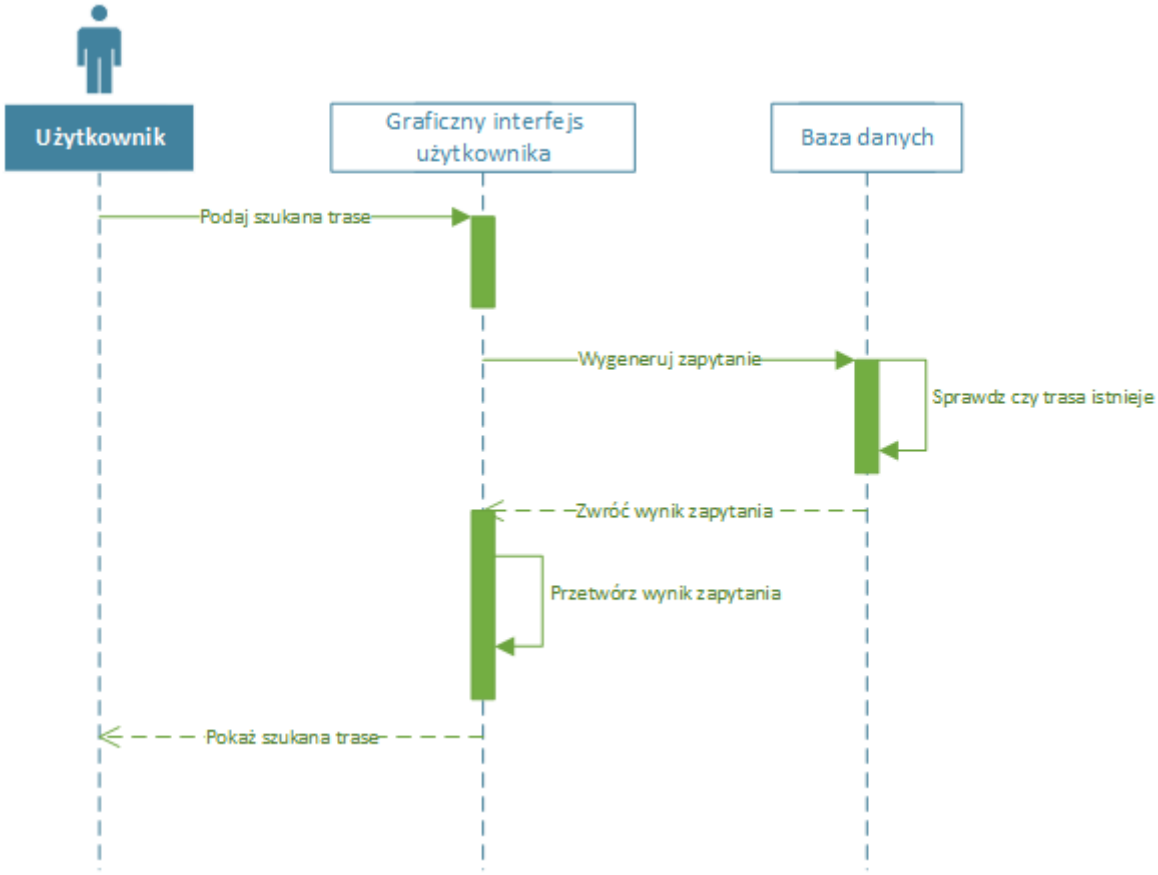
13. Diagram czynności.





14. Diagramy sekwencji.


Administrator
Graficzny interfejs użytkownika
Baza danych
Podaj login/hasło
Wygeneruj zapytanie
Sprawdz czy użytkownik o takim loginie i haśle istnieje
Zwróć wynik zapytania
alt
Jeżeli nie istnieje
Przetwórz wynik zapytania
Zwróć błąd
opt
Jezeli istnieje
Wyświetl graficzny interfejs użytkownika
Usuń rekord
Wygeneruj zapytanie
Przetwórz zapytanie
Zwróć wynik zapytania
Przetwórz wynik zapytania
Wyświetl graficzny interfejs użytkownika
Dodaj rekord
Wygeneruj zapytanie
Przetwórz zapytanie
Zwróć wynik zapytania
Przetwórz wynik zapytania
Wyświetl graficzny interfejs użytkownika



15. Metodologia. Narzędzia CASE.

Mianem narzędzi CASE (ang. Computer Aided Software Engineering) określa się systemy komputerowe, przeznaczone do wspomagania rutynowych czynności procesu tworzenia oprogramowania. To właśnie dzięki nim projekty tworzy się precyzyjniej, a praca nad diagramami, weryfikowanie ich poprawności oraz śledzenie wykonanych testów jest jeszcze szybsze i prostsze.

Narzędzia CASE zastosowane w projekcie:

- ArgoUML – producent University of California, licencja bezpłatna. (http://argouml.tigris.org)
- Eclipse – producent Eclipse Foundation, licencja Eclipse Public License. (http://www.eclipse.org)
- Microsoft Office Excel – producent Microsoft, licencja komercyjna. (http://www.office.microsoft.com)
- Microsoft Office Word – producent Microsoft, licencja komercyjna. (http://www.office.microsoft.com)
- Microsoft SQL Server – producent Microsoft, licencja Microsoft EULA. (http://www.microsoft.com/sql)
- Microsoft SQL Server Management Studio Express – producent Microsoft, licencja Microsoft EULA. (https://www.microsoft.com/en-us/server-cloud/products/sql-server)
- Mozilla Firefox – producent Mozilla Corporation, licencja Mozilla Licensing. (http://www.mozilla.org)

16. Kosztorys.

Poniżej zamieszczono wydatki uwzględnione w 4-miesięcznym procesie wdrażania projektu.

koszty zakupu oprogramowania							
lp.	nazwa	wersja	licencja	cena	ilość	czas realizacji	suma
1	Microsoft Windows	7	komercyjna	599 zł	5	4 mies.	2 995 zł
2	Microsoft Office	365 BP	komercyjna	45 zł / mies.	5		900 zł
3	Eclipse	Java EE Dev.	GPL	-	4		-
4	XAMPP	7.0.4	GPL	-	5		-
5	SAP S. PowerDesigner	16.5	komercyjna	10 467 zł	1		10 467 zł
6	GitHub	Silver	komercyjna	190 zł / mies.	1		760 zł
7	Git Large File Storage	-	komercyjna	19 zł / mies.	1		76 zł
łączny koszt zakupu oprogramowania							**15 198 zł**

koszty pracy				
lp.	zadanie	cena za godz.	czas realizacji	suma
1	analiza wymagań	54 zł	91 godz.	4 914 zł
2	projekt systemu	62 zł	150 godz.	9 300 zł
3	realizacja aplikacji	70 zł	165 godz.	11 550 zł
4	testowanie aplikacji	60 zł	85 godz.	5 100 zł
5	tworzenie dokumentacji	45 zł	90 godz.	4 050 zł
łączny koszt pracy				**34 914 zł**

koszty utrzymania				
lp.	usługa	miesięczny koszt	czas realizacji	suma
1	wynajem lokalu	600 zł	4 mies.	2 400 zł
2	energia elektryczna	400 zł		1 600 zł
3	dostęp do internetu	80 zł		320 zł
4	ogrzewanie	300 zł		1 200 zł
5	dojazdy	600 zł		2 400 zł
6	licencja ZAiKS	81 zł		324 zł
7	komputer (7x)	0 zł (środki własne)		0 zł
8	pracownicy : 2x młodszy programista 2x starszy programista 1x kierownik projektu	12 000 zł / 2x os. 20 000 zł / 2x os. 10 000 zł / os.		168 000 zł
łączny koszt utrzymania				**176 244 zł**

łączny koszt realizacji projektu	**226 356 zł**

17. Narzędzia RAD.

Rapid Application Development (RAD) to metodologia polegająca na udostępnieniu programiście dużych możliwości prototypowania oraz dużego zestawu gotowych komponentów (np. zapewniających dostęp do bazy danych). Umożliwia to uzyskanie pewnego efektu już w pierwszych krokach programistycznych, jednocześnie stanowi poważne zagrożenie dla projektów o większych rozmiarach ze względu na łatwość nieprzemyślanego modyfikowania.

Podczas pracy nad projektem wykorzystane zostało narzędzie Eclipse (framework) do tworzenia aplikacji w języku Java. Narzędzie to zostało wybrane z uwagi na nabyte wcześniej doświadczenie z tym oprogramowaniem.

18. Testy przeprowadzone w ramach projektu.

testy modelu bazy danych

testowany element	środowisko	rezultat
encje	Sybase PowerDesigner 12.5	pozytywny
połączenia między encjami	Sybase PowerDesigner 12.5	pozytywny
klucze	Sybase PowerDesigner 12.5	pozytywny
relacje pomiędzy encjami	Sybase PowerDesigner 12.5	pozytywny
typy danych atrybutów encji	Sybase PowerDesigner 12.5	pozytywny
dodawanie rekordów do tabel	Microsoft SQL Server 2014	pozytywny
generowanie modelu fizycznego bazy danych na podstawie modelu konceptualnego	Sybase PowerDesigner 12.5	pozytywny

testy bazy danych / zgodności ze specyfikacją

nazwa testu	środowisko	warunek spełnienia	rezultat, komentarz
model konceptualny – zachowanie logiki i zgodność ze specyfikacją wymagań	Sybase PowerDesigner 12.5	• obecność tabeli „AUTOKARY”; • obecność tabeli „KIEROWCY”; • obecność tabeli „KURSY”; • poprawnie przypisane typy danych do atrybutów encji;	pozytywny
wygenerowanie modelu	Sybase PowerDesigner 12.5	• stworzenie	negatywny – w

fizycznego na podstawie modelu konceptualnego		działającego modelu fizycznego;	tabeli „TRASA” atrybut „TR_Dzien_tyg” ma nieprzypisany typ danych
model fizyczny – poprawność przypisania kluczy	Sybase PowerDesigner 12.5	• każda tabela musi posiadać klucz podstawowy; • odpowiednio przypisane klucze obce;	pozytywny
wygenerowanie bazy danych na podstawie modelu fizycznego (poprawionego)	Sybase PowerDesigner 12.5	• wygenerowanie kodu SQL;	pozytywny
baza danych – dodawanie rekordów do tabel	Microsoft SQL Server 2014	• dodanie pojedynczego rekordu do każdej tabeli;	pozytywny

testy gotowego produktu

nazwa testu	warunki spełnienia	rezultat testu, komentarz
GUI – menu główne	- możliwość przejścia do rozkładu jazdy - możliwość przejścia do logowania	pozytywny
GUI – rozkład jazdy	- opcja powrotu do menu głównego - możliwość wyboru miejsca startowego - możliwość wyboru miejsca docelowego - poprawne wyświetlenie kursów odpowiadających kryterium wyszukiwania	pozytywny
GUI – okno logowania	- możliwość wpisania loginu - możliwość wpisania hasła - obecność guzika „zaloguj”	pozytywny, brak informacji o wpisaniu błędnego loginu lub hasła, hasło jest niezakodowane
GUI – trasy	- wyświetlenie wcześniejszych rekordów z tabeli TRASA	pozytywny

	- dodanie rekordu do tabeli - usunięcie rekordu z tabeli	
GUI – przystanki	- wyświetlenie wcześniejszych rekordów z tabeli PRZYSTANKI - dodanie rekordu do tabeli - usunięcie rekordu z tabeli	pozytywny
GUI – kierowcy	- wyświetlenie wcześniejszych rekordów z tabeli KIEROWCY - dodanie rekordu do tabeli - usunięcie rekordu z tabeli	pozytywny
GUI – autokary	- wyświetlenie wcześniejszych rekordów z tabeli AUTOKARY - dodanie rekordu do tabeli - usunięcie rekordu z tabeli	pozytywny
GUI – kursy	- wyświetlenie wcześniejszych rekordów z tabeli KURSY - dodanie rekordu do tabeli - usunięcie rekordu z tabeli	pozytywny
GUI – miejscowości	- wyświetlenie wcześniejszych rekordów z tabeli MIEJSCOWOSCI - dodanie rekordu do tabeli - usunięcie rekordu z tabeli	pozytywny
GUI – rejestr przejazdów	- wyświetlenie wcześniejszych rekordów z tabeli REJESTR_PRZEJAZDOW - dodanie rekordu do tabeli - usunięcie rekordu z tabeli	pozytywny

DOKUMENT DETALICZNY PROJEKTU
dla firmy transportowej

wersja 1.0

1. Streszczenie.

Niniejszy dokument detaliczny projektu (DDP) prezentuje szczegóły pracy zespołu projektowego, nad stworzeniem aplikacji bazodanowej wspomagającej organizację pracy w firmie transportowej. Pierwsza część dokumentu zawiera opis ogólnych założeń projektowych, druga zaś opisuje wykorzystane w projekcie komponenty.

SPIS TREŚCI

2. Wprowadzenie.

2.1. Cel.

Dokument detaliczny projektu ma za zadanie przedstawić szczegółowo sposób realizowanych prac. Zgromadzone są w nim wszystkie informacje odnośnie budowy i działania oprogramowania. Określa on również założenia projektu, standardy, narzędzia i komponenty wchodzące w skład aplikacji.

2.2. Zakres.

Założeniem projektu jest stworzenie bazodanowej aplikacji *DamberPol*, służącego do wspomagania organizacji pracy w firmie transportowej. System ma służyć do przechowywania różnego rodzaju informacji, m.in. rozkładu jazdy. Aplikacja pozwoli na magazynowanie informacji o kierowcach, pasażerach oraz pracownikach. Umożliwi pasażerom przeglądanie rozkładu jazdy.

2.3. Definicje, akronimy, skróty.

- Pasażer – niezarejestrowany osoba firmy transportowej, nieposiana własnego loginu i hasła do systemu, posiada możliwość przeglądania zasobów rozkładu jazdy.
- Pracownik – pracownik firmy, posiada własny identyfikator i hasło do logowania do systemu, posiada możliwość przeglądania zasobów (raporty, rozkłady jazdy, kierowcy zatrudnieni w firmie itp.), dodawania, edytowania już istniejących pozycji.

2.4. Odsyłacze.

Ustawa z dn. 29 sierpnia 1997 r. o ochronie danych osobowych (Dz. U. 1997 nr 133 poz. 883 z późn. zm.).

2.5. Opracowanie.

Wyżej wymieniony dokument powstał na bazie specyfikacji wymagań systemu określonych podczas pierwszych etapów projektowania systemu. Zawiera on definicje standardów, które będą przestrzegane podczas realizacji projektu. Dalsza część dokumentu zawiera informacje o narzędziach, modułach i komponentach systemu oraz interfejsie graficznym oprogramowania.

3. Projekt. Standardy, konwencje, procedury.

3.1. Standardy projektowe.

Z uwagi na niewielkie doświadczenie zespołu dot. prac projektowych, wykorzystaliśmy tzw. model przyrostowy tworzenia oprogramowania. Model ten gwarantuje częste kontakty z klientem, brak konieczności zdefiniowania z góry całości wymagań, wczesne wykorzystanie przez klienta fragmentów systemu, możliwość elastycznego reagowania na opóźnienia realizacji fragmentu – przyspieszenie prac nad inną / innymi częściami. Wyżej wymienione punkty sprawiają, że ryzyko porażki prac nad całym projektem jest dużo mniejsze.

3.2. Standardy dokumentacyjne.

Wszystkie dokumenty projektu stworzone zostały na podstawie jednego firmowego szablonu (papieru firmowego). W trakcie tworzenia dokumentacji zastosowany został jednolity, przejrzysty język, który jest wspólny dla całego dokumentu. Podczas programowania aplikacji zostały

zastosowane komentarze – powodują one, że kod oprogramowania jest bardziej zrozumiały, a także umożliwiają łatwą edycję kodu aplikacji nawet przez osoby niezwiązane z pracą projektową.

3.3. Konwencje nazewnicze.

Zastosowane w projekcie słownictwo jest ukierunkowane na prostotę i jednoznaczność. W implementacji została zachowana konwencja nazewnictwa Java.

3.4. Standardy programistyczne.

System został zaprojektowany i stworzony w oparciu o język programowania Java. W projekcie będzie wykorzystywane podejście obiektowe do programowania oraz wykorzystywany będzie wzorzec projektowy MVC (Model-View-Controller). Aplikacja będzie stworzona zgodnie z wcześniejszymi założeniami i wcześniej ustalonym diagramem klas. Zaletami takiego podejścia jest brak zależności modelu od widoków aplikacji oraz łatwość dodawania i modyfikowania istniejących widoków bez wpływu na kluczową część systemu.

3.5. Narzędzia.

Do zrealizowania oprogramowania wykorzystano język Java. Środowisko RAD w projekcie zastosowano w postaci aplikacji Eclipse. Podczas tworzenia dokumentacji posługiwaliśmy się oprogramowaniem StarUML i oprogramowaniem firmy Microsoft Office 2007. Do stworzenia bazy danych wykorzystaliśmy narzędzie „PowerDesigner" firmy Sybase.

Pełna lista narzędzi wykorzystywanych podczas pracy nad systemem i dokumentacją znajduje się poniżej.

- ArgoUML – producent University of California, licencja bezpłatna. (http://argouml.tigris.org)
- Eclipse – producent Eclipse Foundation, licencja Eclipse Public License. (http://www.eclipse.org)
- Microsoft Office Excel – producent Microsoft, licencja komercyjna. (http://www.office.microsoft.com)
- Microsoft Office Word – producent Microsoft, licencja komercyjna. (http://www.office.microsoft.com)
- Microsoft SQL Server – producent Microsoft, licencja Microsoft EULA. (http://www.microsoft.com/sql)
- Microsoft SQL Server Management Studio Express – producent Microsoft, licencja Microsoft EULA. (https://www.microsoft.com/en-us/server-cloud/products/sql-server)
- Mozilla Firefox – producent Mozilla Corporation, licencja Mozilla Licensing. (http://www.mozilla.org)

4. Specyfikacja komponentów.

4.1. Klasa "Autokary".

Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Autokary.

4.2. Klasa “Kierowcy”.
Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Kierowcy.

4.3. Klasa “Kursy”.
Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Kursy.

4.4. Klasa “Miejscowosci”.
Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Miejscowosci.

4.5. Klasa “Przystanki”.
Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Przystanki.

4.6. Klasa “Rejestr_przejazdow”.
Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Rejestr_Przejazdow.

4.7. Klasa “Rozklad_Jazdy”.
Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Autokary.

4.8. Klasa “Trasa”.
Klasa zawiera settery oraz gettery, które umożliwiają uzupełnienie tabel wartościami pobranymi z bazy danych z encji Autokary.

4.9. Klasa “checkbox”.
Klasa odpowiadająca za wyświetlanie dodatkowego okna podczas próby usunięcia rekordu.
Okno ma za zadanie potwierdzić chęć usunięcia rekordu przez użytkownika.

4.10. Klasa “Loguj”.
Klasa odpowiedzialna za wyświetlenie okna, które pozwala na zalogowanie się do aplikacji poprzez wpisanie poprawnego loginu oraz poprawnego hasła.

4.11. Klasa “Main”.
Główna klasa odpowiedzialna za cale działanie aplikacji.

4.12. Klasa “ConnectionToDB”.
Klasa odpowiedzialna za połączenie z bazą danych za pomocą poprawnie skonfigurowanego sterownika sqljdbc4.

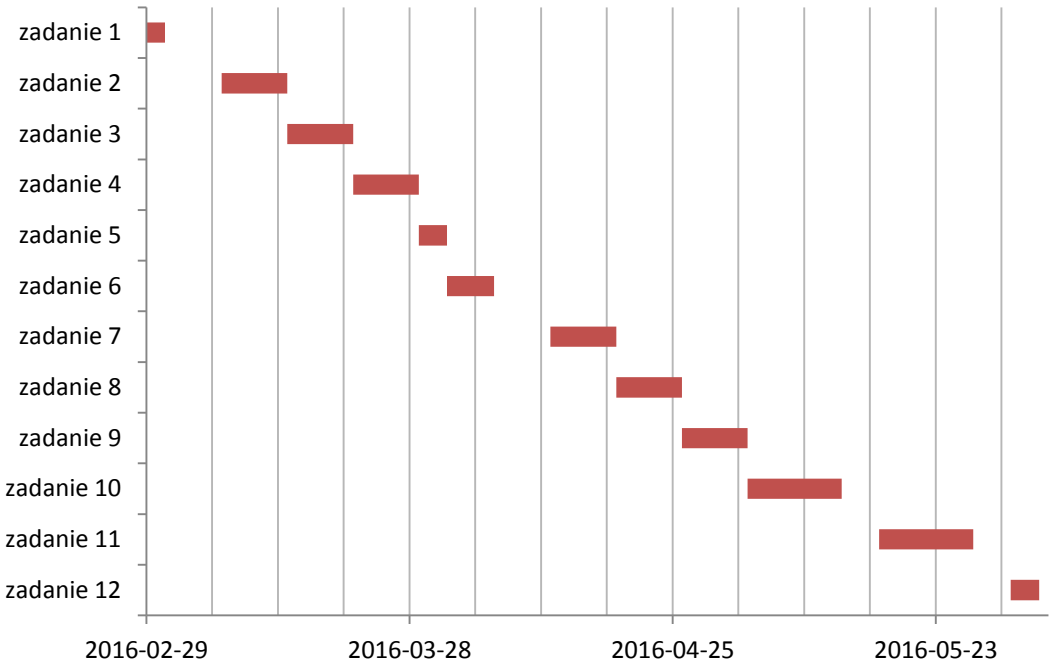
4.13. Klasa ”FillingTables”.
Klasa odpowiedzialna za wypełnianie tabel po wcześniejszym wywołaniu metody odpowiedzialnej za połączenie z bazą danych.

4.14. Klasa ”Queries”.
Klasa zawierająca metody, które realizują zapytania do bazy danych jak np. pobranie całej tabeli, wprowadzenie do niej danych.

4.15. Biblioteka “sqljdbc4”.
Biblioteka odpowiedzialna za możliwość połączenia z bazą danych SQL Server.

5. Załączniki.

5.1. Harmonogram prac.



Zadanie 3 (15.03 – 22.03). Zaprojektowanie modelu bazy danych (model konceptualny i model fizyczny), stworzenie bazy danych na potrzeby projektu.

Zadanie 6 (01.04 – 06.04). Zaprojektowanie graficznego interfejsu użytkownika. Opracowanie słownika danych zawierający specyfikację modelu.

Zadanie 8 (19.04 – 26.04). Wybór odpowiednich narzędzi do szybkiego rozwijania aplikacji (ang. RAD). Ciąg dalszy prac nad projektem (zaprogramowanie GUI, podstawowych funkcjonalności).

Zadanie 9 (26.04 – 03.05). Prace nad programowaniem dalszych funkcjonalności systemowych.

Zadanie 10 (03.05 – 13.05). Wstępne testowanie aplikacji. Naprawa ewentualnych błędów.

Zadanie 11 (17.05 – 27.05). Zakończenie programowania projektu. Przeprowadzenie szczegółowych testów. Finalizacja projektu. Sprawdzenie i zamknięcie dokumentacji. Przygotowanie projektu do zaprezentowania klientowi.

Zadanie 12 (31.05 – 03.06). Prezentacja projektu klientowi. Zakończenie prac nad projektem.

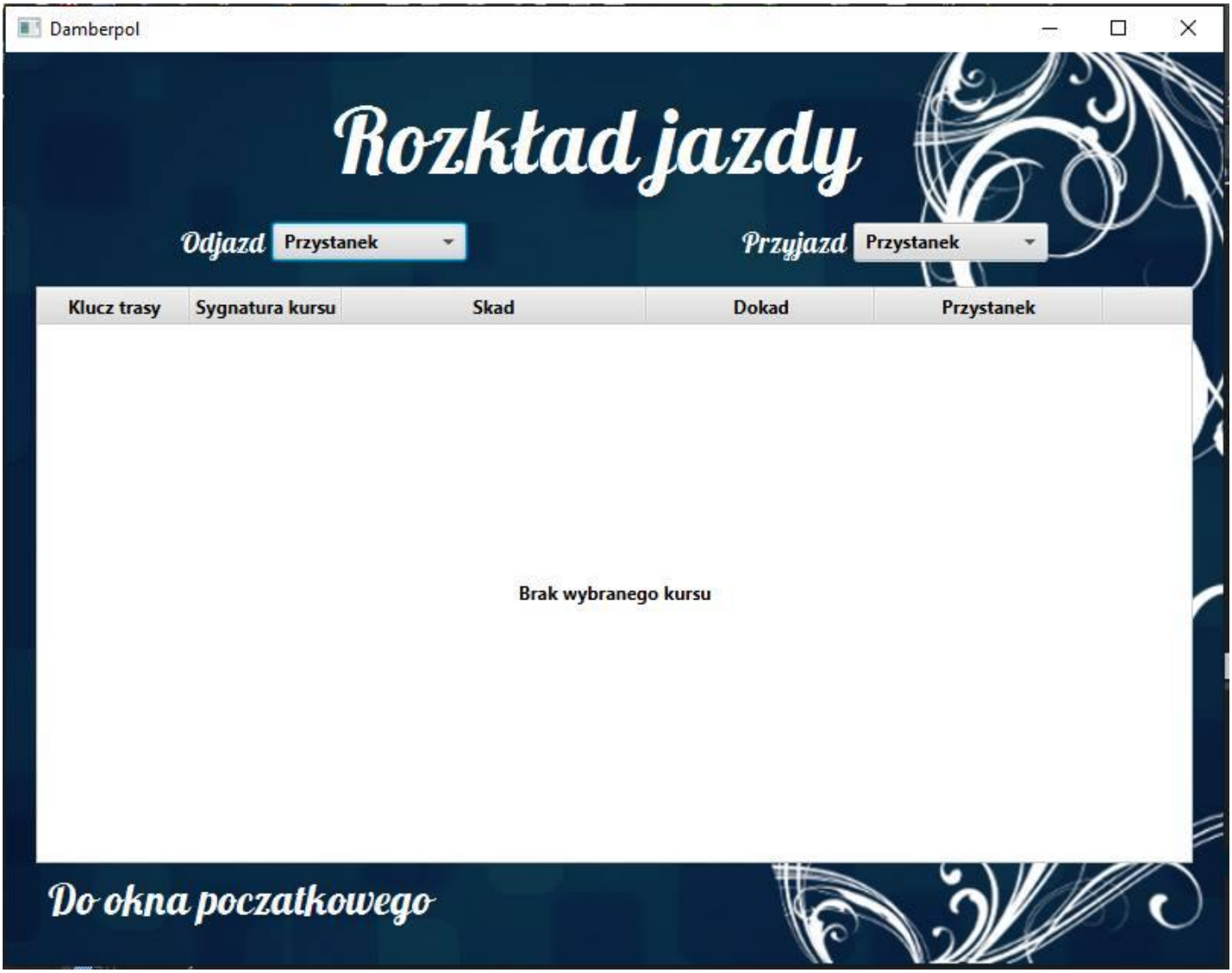
5.2. Interfejs aplikacji.

5.2.1. Okno powitalne.

5.2.2. Logowanie do aplikacji.

5.2.3. Aplikacja.

5.2.4. Rozkład jazdy.



5.2.5. Okno dialogowe.