System obsługi sklepu rowerowego

Spis treści

[Opis systemu 1](#_Toc156235897)

[Backend 1](#_Toc156235898)

[Diagram klas 2](#_Toc156235899)

[Obsługa Danych 2](#_Toc156235900)

[Interface IDatabase 2](#_Toc156235901)

[Obsługa produktów 3](#_Toc156235902)

[Klasa Bike 3](#_Toc156235903)

[Klasa KidsBike 7](#_Toc156235904)

[Klasa Database\_bikes 8](#_Toc156235905)

[Obsługa klientów 9](#_Toc156235906)

[Klasa Client 10](#_Toc156235907)

[Klasa Company 10](#_Toc156235908)

[Klasa Private\_person 11](#_Toc156235909)

[Klasa Database\_client 12](#_Toc156235910)

[Obsługa transakcji 13](#_Toc156235911)

[Klasa Transaction 13](#_Toc156235912)

[Klasa Database\_transaction 16](#_Toc156235913)

[Obsługa Błędów 17](#_Toc156235914)

[Obsługa baz danych SQL 17](#_Toc156235915)

[Testy jednostkowe 19](#_Toc156235916)

# Opis systemu

System jest zaprojektowany do użytku przez pracowników sklepu rowerowego. Ma na celu obsługę trzech kategorii działalności sklepu : jego klientów , produktów jakie oferuje( w tym przypadku rowerów ) oraz transakcji jakie zawiera. Dzięki temu systemowi pracownik jest w stanie szybciej i sprawniej zapisywać i zmieniać dane dotyczące powyżej wymienionych trzech kategorii działalności sklepu.

# Backend

Backend został napisany jako aplikacja konsolowa na platformie NET 7.0 w języku C#.

## Diagram klas

Na poniżej załączonym diagramie klas, zobrazowana jest zawartość całego projektu.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

## Obsługa Danych

### Interface IDatabase

W celu ujednolicenia metod znajdujących się w klasach zwierających dane, powstał interface IDatabase.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Metoda GetInfo ma pozwalać na znalezienie obiektu za pomocą jego klucza w słowniku, a metoda ZapiszXML ma umożliwić zserializowanie danych.

Do interface’u są napisane dwie metody rozszerzające:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Statyczna klasa public static class DatabaseExtensions posiada metody:

Statyczna metoda public static string DictionarieToString<T>(this IDatabase database, Dictionary<string, T> dictionary) konwertuje zawartość każdego słownika (Dictionary<string, T>) na ciąg znaków, używając metody ToString() dla każdej wartości słownika.

Statyczna metoda public static string ListToString<T>(this IDatabase database, List<T> list) konwertuje zawartość każdej listy (List<T>) na ciąg znaków, używając metody ToString() dla każdego elementu listy. Jeżeli lista jest pusta zwracany jest odpowiedni komunikat.

## Obsługa produktów

Z odsługa produktów w projekcie mamy powiązane 3 klasy:

* *Bike*
* *KidsBike*
* *Database\_bikes*

Gdzie *KidsBike* dziedziczy po *Bike* oraz klasa *Database\_bikes* przechowuje obiekty dwóch wcześniej wymienionych klas w słownikach.

### Klasa Bike

Klasa *Bike* implementuje interfejst IComparable oraz IEquatable.

Publiczne pola klasy:

public string model;

public int size;

public double price;

public string id\_bike;

public DateTime storingDate;

oraz pola statyczne:

public static int index;

public static double minimalPrice;

Dodatkowo posiada dwa publiczne pola typu wyliczeniowego: public EnumCompany company umożliwiające wybór firmy roweru spośród: Trek, Speciallized, Cube, Merida, Bianchi oraz public EnumType type dotyczące typu roweru: MT , Road , Cyclocross.

Wszystkie pola posiadają wygenerowane właściwości. W przypadku właściwości Size za pomocą wirtualnej metody bool Sizing(int x) dokonywane jest sprawdzenie poprawności rozmiaru roweru w zależności od jego typu. W przypadku nieprawidłowego rozmiaru wyrzucony zostaje wyjątek InValidDataException z odpowiednim komunikatem.

Jak działa metoda *Sizing* ?

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Jako argument przyjmuje typ intiger , który ma odzwierciedlać rozmiar ramy rowerowej. Korzystając z *switch*/*case* , metoda sprawdza czy wprowadzone przez pracownika dane pokrywają się z realiami. Różne typy rowerów mają różne zakrey rozmiarowe ram, które zostały zawarte w warunkach. Metoda jest wirtualna, ponieważ nadpisujemy ją w klasie *KidsBike* zmieniając zakresy rozmiarów.

Dla właściwości Price dokonywane jest sprawdzenie, czy cena roweru jest wyższa od ceny minimalnej. Jeśli nie zostało to spełnione, wyrzucony zostaje wyjątek z komunikatem "Price should be at least 100 zł" informującym o minimalnej cenie wypożyczanego roweru.

Konstruktor statyczny inicjuje pole index na wartość 0 oraz MinimalPrice na wartość 100.

Konstruktor nieparametryczny zwiększa liczebność pola index o 1.

Konstruktor parametryczny inicjuje wszystkie pola oraz wykorzystuje konstruktor nieparametryczny.

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, zrzut ekranu

Opis wygenerowany automatycznie

Data wypożyczenia roweru storingDate przekazywana jest jako tekstowy string, a następnie konwertowana na tym daty DateTime. Właściwość price jest ustawiana na podstawie wyniku metody *Pricing* oraz Id\_bike jest ustawiane na podstawie wyniku metody ID()

Jak działa metoda *Pricing*?

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Publiczna metoda double Pricing(double basePrice) ustala cenę roweru na podstawie ceny podstawowej oraz liczby dni od daty przechowywania. Zmienna double daysSinceStoring określa liczbę dni, na jaką rower został wypożyczony. Jeżeli wypożyczenie jest dłuższe niż rok, do ceny wypożyczenia roweru naliczany jest 15% rabat.

Jak działa metoda *ID*?

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Publiczna metoda wirtualna virtual string ID() ustala indywidualny identyfikator roweru na podstawie wprowadzonego typu. Wykorzystuje do tego celu instrukcję switch. Dzięki tak napisanej metodzie i konstruktorze nieparametrycznym który zwieksza pole indkes o jeden po każdym dodaniu obiektu, mamy pewność że Id rowerów będą unikatowe.

Implementacja interface IEquatable :

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Metoda *Equals* zwraca typ logiczny bool, po porównaniu ze sobą modeli rowerów. Metodę tą wykorzystuje w metodzie ComapreTo.

Implementacja interface IComparable :

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Najpierw sprawdzamy czy obiekty nie są puste. Nastepnie porównanie ze sobą Marek rowerów. Jeżeli są takie same, porównujemy ze sobą modele rowerów (funkcja *Equals*)

Wypisywanie danych na temat rowerów obsługują 2 funkcje:

* Nadpisana metoda *ToString*, która wypisuje wszystkie właściowości roweru
* Wirtualna metoda *ShortString*, która wypisuje markę, model, typ, i rozmiar roweru

### Klasa KidsBike

Klasa KidsBike jest klasą dziedziczącą po klasie Bike oraz implementuje interface IComparable. Nie zawiera żadnych dodatkowych pól w stosunku do klasy *Bike*.

Konstruktor nieparametryczny oraz konstruktor parametryczny wykorzystujący konstruktor parametryczny z klasy bazowej. Do pola Id\_bike przypisana jest nadpisana metoda ID(), która dodaje literę "K".

Nadpisana metoda public override bool Sizing(int x) działa tak jak w klasie *Bike* tylko ma inne zakresy rozmiarów ram.

Nadpisana metoda *ToString* oraz *ShortString* wypisuje podstawowe informacje dotyczące wybranego roweru uwzględniając fakt że jest to dziecięcy model.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Implementacja IComparable

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

W tej klasie nie implementowany jest iinterface IEquatalble, dlatego do porównywania modeli służy string.Compare. Sygnatura metody string.Compare wygląda następująco:

**public static int Compare(string strA, string strB, StringComparison comparisonType);**

Comparison type = Ordinal , oznacza że ciagi znaków sa porównywane zgodnie z kodem porządkowym W tym przypadku, porównywanie jest oparte na wartości numerycznej kodu ASCII dla każdego znaku.

### Klasa Database\_bikes

Klasa Database\_bike implementuje interface IDatabase

Klasa Database\_bikes reprezentuje bazę danych rowerów i rowerów dziecięcych. Klasa umożliwiaja dodawanie, usuwanie i pobieranie obiektów rowerów na podstawie różnych kryteriów. Poniżej znajduje się przegląd klasy, jej właściwości i metody:

Pola:

string databasename;

Dictionary<string, Bike> bikes = new Dictionary<string, Bike>();

Dictionary<string, KidsBike> kidsbikes = new Dictionary<string, KidsBike>();

List<Bike> bikesList = new List<Bike>();

List<KidsBike> kidsbikesList = new List<KidsBike>();

Konstruktor nieparametryczny oraz konstruktor parametryczny inicjujący nazwę bazy danych na daną wartość.

Publiczna metoda public void AddObject<T>(T obj) where T : Bike dodaje obiekt roweru (Bike lub KidsBike) do odpowiedniego słownika na podstawie jego typu. Jeżeli dany obiekt nie należy do klasy Bike ani Kidsbike wyrzucony zostaje wyjątek InValidDataException. Litera T w argumrncie funkcji reprezentuje Typ obiektu który jest dodawany do słownika /listy. Aby metoda działała dla każdego typu roweru jaki chcemy dodać ( zwykły lub dziecięcy ), musimy dodać typ T gdzie T to Bike( po którym dziedziczy KidsBike więc też jest Bike).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Publiczna metoda public void RemoveObject<T>(T obj) where T : Bike usuwa obiekt roweru (Bike lub KidsBike) z odpowiedniego słownika na podstawie jego typu. Jeżeli dany obiekt nie należy do klasy Bike ani Kidsbike wyrzucony zostaje wyjątek InValidDataException.

Wyszukiwanie rowerów:

Publiczna metoda public string GetInfo(string key) pobiera i zwraca informacje o rowerze lub rowerze dziecięcym na podstawie jego unikalnego klucza. Jeśli klucz nie zostanie znaleziony, zwracany jest odpowiedni komunikat.

Publiczna metoda public List<Bike> SearchBikeName(EnumCompany name) wyszukuje rowery (zarówno zwykłe, jak i dziecięce) według nazwy firmy. Zwraca listę pasujących rowerów lub odpowiedni komunikat, jeśli żadne nie zostaną znalezione.

Publiczna metoda public List<Bike> SearchBikeSize(int size) wyszukuje rowery (zarówno zwykłe, jak i dziecięce) według rozmiaru. Zwraca listę pasujących rowerów lub odpowiedni komunikat, jeśli żadne nie zostaną znalezione.

Metoda rozszerajaca *ListToString* umożliwia wypisanie wyników wyszukiwania w konsoli. Np.

**Console.WriteLine(product.ListToString(product. SearchBikeSize (47)));**

Nadpisana metoda ToString() wypisuje wszystkie rowery oraz rowery dziecięce znajdujące się w bazie danych.

Ponad to jest możliwość zserializowania i deserializacji pliku z danymi o rowerach za pomocą poniższych funkcji:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

## Obsługa klientów

Z obsługa klientów w projekcie mamy powiązane 4 klasy

* *Client*
* *Private\_person*
* *Company*
* *Database\_clients*

Z czego klasa *Client* jest klasa abstrakcyjną ( oznacza to że nie można stworzyć obiektu *Client* ) i dziedziczą po niej *Private\_person* oraz *Company*. Klasa *Database\_clients* przechowuje obiekty powyższych klas w słownikach.

### Klasa Client

Klasa Client posiada publiczne pola:

public string id\_client;

public string country;

public string city;

public string street;

public string code;

pole prywatne:

private string pattern = "^\\d{2}-\\d{3}";

oraz pole statyczne:

private static int index;

Wszystkie pola posiadają wygenerowane właściwości.

Publiczna metoda public bool CheckCode(string code, string sPattern) sprawdza przy pomocy wyrażeń regularnych z klasy Regex, czy wprowadzony kod jest zgodny z wymaganym wzorem. Zwracana jest wówczas wartość true, w przeciwnym wypadku zwracana jest wartość false.

Konstruktor statyczny inicjuje pole index na wartość 1.

Konstruktor nieparametryczny zwiększa liczebność pola index o 1 oraz inicjuje pole Id\_client na pusty napis.

Konstruktor parametryczny inicjuje wszystkie pola oraz wykorzystuje konstruktor nieparametryczny. W przypadku pola Code dokonuje sprawdzenia poprawności kodu wykorzystując metodę public bool CheckCode(string code, string sPattern). Gdy kod jest zgodny z wymaganym wzorem do pola Code zostaje przypisana odpowiednia wartość. W przeciwnym wypadku wyrzucony zostaje wyjątek InValidDataException z komunikatem "Post code must be inscribe according to this pattern: {00}-{000} , where zeros reprezents numbers" informującym o niepoprawnym formacie kodu.

Nadpisana metoda ToString() wypisuje podstawowe informacje o kliencie.

### Klasa Company

Klasa Company jest klasą dziedziczącą po klasie Client. Zawiera dodatkowe pola publiczne:

public string companyName;

public string nip;

Dodatkowe pola posiadają wygenerowane właściwości. W przypadku właściwości nip w metodzie init dokonywane jest sprawdzenie, czy nip składa się z 10 lub 11 cyfr. W przeciwnym wypadku wyrzucany jest wyjątek InValidDataException z komunikatem "Nip should be 11 or 10 chracters long" informującym o nieodpowiedniej liczbie cyfr w numerze nip. Ta ewidencja bazuje na regulacjach w polskim prawie, na temat numerów nip.

Konstruktor nieparametryczny wykorzystuje konstruktor z klasy bazowej.

Konstruktor parametryczny inicjuje pola na określone wartości i wykorzystuje konstruktor parametryczny z klasy bazowej. Do pola Id\_client zostało przypisane pole Id\_client z klasy bazowej wraz z literą "C".

Nadpisana metoda ToString() wypisuje podstawowe informacje o kliencie uwzględniając nazwę firmy oraz numer nip.

Implementacja IComparable

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Firmy są sortowane na podstawie ich nazw, jeżeli sa takie same to dokonywane jest sortowanie po państwach z jakich pochodzą.

Implementacja IClonable

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Metoda Clone , zwraca płytki klon obiektu.

### Klasa Private\_person

Klasa Private\_person jest klasą dziedziczącą po klasie Client.

Posiada publiczne pola:

public string name;

public string surname;

public EnumGender gender;

Wszystkie pola posiadają wygenerowane właściwości.

Konstruktor nieparametryczny wykorzystuje konstruktor z klasy bazowej.

Konstruktor parametryczny inicjuje pola na określone wartości i wykorzystuje konstruktor parametryczny z klasy bazowej. Do pola Id\_client zostało przypisane pole Id\_client z klasy bazowej wraz z literą "P".

Nadpisana metoda ToString() wypisuje podstawowe informacje o kliencie uwzględniając imię, nazwisko oraz płeć klienta.

Implementacja IComparable

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Publiczna metoda public int CompareTo(Private\_person? other) porównuje osoby prywatne po nazwisku. W przypadku, gdy nazwiska są takie same porównywane są imiona.

Implementacja IClonable

Wygląda tak samo jak w klasie *Company.*

### Klasa Database\_client

Klasa Database\_clients reprezentuje bazę danych klientów, którzy chcą wypożyczyć dany rower.

Klasa posiada publiczne pole prywatne:

public string databasename;

publiczne słowniki:

public Dictionary<string, Company> companies = new Dictionary<string, Company>();

public Dictionary<string, Private\_person> persons = new Dictionary<string, Private\_person>();

oraz publiczne listy:

public List<Company> companiesList = new List<Company>();

public List<Private\_person> personsList = new List<Private\_person>();

Konstruktor nieparametryczny oraz konstruktor parametryczny wywołujący konstruktor nieparametryczny i inicjujący nazwę bazy danych na daną wartość.

Publiczna metoda public void AddObject<T>(T obj) where T : Client dodaje obiekt klienta (Company lub Private\_person) do odpowiedniego słownika i listy na podstawie jego typu. Jeżeli dany obiekt nie należy do klasy Company ani Private\_person wyrzucony zostaje wyjątek InValidDataException.

Publiczna metoda public void RemoveObject<T>(T obj) where T : Client usuwa obiekt klienta (Company lub Private\_person) do odpowiedniego słownika i listy na podstawie jego typu. Jeżeli dany obiekt nie należy do klasy Company ani Private\_person wyrzucony zostaje wyjątek InValidDataException.

Publiczna metoda public string GetInfo(string key) pobiera i zwraca informacje o firmie lub osobie prywatnej na podstawie jej unikalnego klucza. Jeśli klucz nie zostanie znaleziony, zwracany jest odpowiedni komunikat.

Publiczna metoda public List<Company> SearchCompanyName(string name) wyszukuje firmy według nazwy. Zwraca listę pasujących nazw lub odpowiedni komunikat, jeśli żadne nie zostaną znalezione.

Publiczna metoda public List<Private\_person> SearchPersonSurname(string name) wyszukuje klienta będącego osobą prywatną po nazwisku. Zwraca listę pasujących klientów lub odpowiedni komunikat, jeśli żaden nie zostanie znaleziony.

Publiczna metoda public void SortPeople() sortuje osoby prywatne za pomocą funkcji sort, która korzytsa z ustawionego przez nas komparatora omawianego w klasie Private\_person.

Publiczna metoda public void SortCompanies() sortuje osoby prywatne za pomocą funkcji sort, która korzytsa z ustawionego przez nas komparatora omawianego w klasie Company.

Nadpisana metoda ToString() wypisuje wszystkie firmy oraz osoby prywatne znajdujące się w bazie danych.

Za pomocą funcki ZapiszXML oraz OdczytXml istnieje możliwość serializacji i deserailizacji danych. Funkcje działają tak samo jak w klasie *Database\_bikes.*

Implemntacja IClonable

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

## Obsługa transakcji

Z obsługą transakcji mamy powiazane 2 klasy w projekcie :

* *Transaction*
* *Database\_transaction*

Klasa Database\_transaction przechowuje obiekty klasy transaction w słowniku.

### Klasa Transaction

Klasa Transaction odpowiada za zakupy dokonane przez klientów.

Klasa posiada pola prywatne:

private static int index;

private string id\_tran;

private Client client;

private List<object> products = new List<object>();

private bool invoice;

private DateTime date;

Lista products odpowiada za przechowywanie produktów jakie zakupili klienci.

Pola posiadają wygenerowane właściwości.

Konstruktor statyczny ustawia pole indeks na wartość 100.

Konstruktor nieparametryczny zwiększa wartość pola indeks o 1.

Konstruktor parametryczny inicjuje pola na określone wartości i wykorzystuje konstruktor nieparametryczny z tej samej klasy. Data wprowadzana jest jako napis, a następnie konwertowana na typ DateTime.

Publiczna metoda public void AddProduct(object product, Database\_bikes dataset) dodaje rower z wybranej bazy danych do transakcji. Istnieją dwa zabezpieczenia. Po pierwsze metoda SellBike, która po Id roweu sprawdza czy jest on na magazynie oraz sprawdzenie czy data transakcji nie „kłóci się” z data wprowadzenia roweru do magazynu. W przypadku nie spełnienia kryteriów na konsoli pojawia się komunikat.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Publiczna metoda public double SumT() zwraca sumę transakcji.

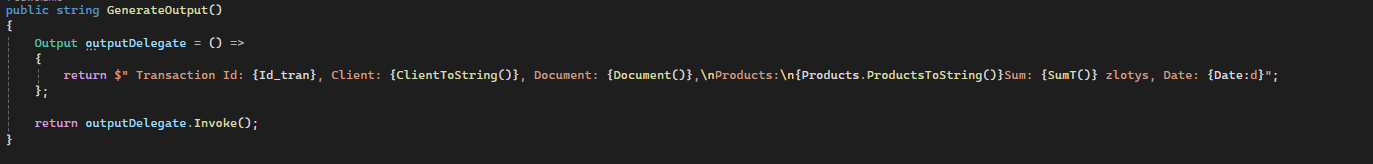
Publiczna metoda public string Document() wykorzystuje pole bool invoice, w przypadku gdy jest prawdziwe zwraca string „invoice” , w przeciwnym razie zwraca string „Receipt”.

Obraz zawierający tekst, Czcionka, zrzut ekranu, linia

Opis wygenerowany automatycznie

Publiczna metoda public string ClientToString() zwraca napis przypasowany do tego czym jest obiekt Client.

Publiczna metoda public string GenerateOutput() przyuzyciu delegatu public delegate string Output(), tworzy stringa wypisanej transakcji.



Wewnatrz tej metody używane są wcześniej opisane funkcje oraz metoda rozszerzającą public static string ProductsToString(this List<object> Products) która działa w następujący sposób:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

GroupBy: Metoda ta grupuje elementy listy na podstawie ich wartości. W tym przypadku, grupuje elementy listy Products według ich wartości.

Select: Dla każdej grupy, tworzy nowy anonimowy obiekt zawierający Product (klucz grupy, czyli wartość elementu) i Count (ilość elementów w grupie).

Jeśli ilość elementów w grupie (groupedProduct.Count) jest większa niż 1, oznacza to, że mamy kilka takich samych produktów. Wtedy do wynikowego StringBuilder dodawana jest linia zawierająca ilość i krótki opis produktu (ShortString), który jest dynamicznie rzutowany z obiektu na odpowiedni typ i uzyskiwany za pomocą metody ShortString

Jeśli ilość elementów w grupie wynosi 1, oznacza to, że mamy tylko jeden taki produkt. Wtedy do wynikowego StringBuilder dodawana jest linia z krótkim opisem tego produktu.

Na koniec, metoda zwraca cały wynik jako jeden ciąg tekstowy, który zawiera informacje o produktach w formie zgrupowanej.

Przykładowo, jeśli Products zawiera różne obiekty, ta metoda grupuje je według ich wartości, liczy ilość każdego z nich i zwraca wynik w formie tekstu, gdzie dla wielokrotnie występujących produktów podana jest ich ilość. Czyli, przykładowo, "2 x ProduktA, 3 x ProduktB".

Ponad to, aby umozliwic serializacje klasy Database\_transaction, nad klasą transaction musimy uwzględnić XmlInclude, ponieważ obiekty transakcja przechowują obiekty Klientów oraz produktów.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

### Klasa Database\_transaction

Klasa database\_transaction przechowuje dane na temat transakcji w słowniku oraz liście.

Posiada pola prywatne

private Dictionary<string, Transaction> transactions = new Dictionary<string, Transaction>();

private string databasename;

private List<Transaction> transactionsList = new List<Transaction>();

Do pól są wygenerowane właściwości.

Klasa posiada konstruktor nieparametryczny oraz konstruktor parametryczny ustawiający nazwę bazy danych.

Publiczna metoda public void AddTran(Transaction tran) dodaje transakcje do słownika i listy.

Publiczna metoda public void RemoveTran(Transaction tran) usuwa transakcje ze słownika i listy.

Publiczna metoda public string GetInfo(string key), wyszukuje transakcje po jej id.

Publiczna metoda public List<Transaction> SearchClient(string clientName), zwraca listę klientów których nazwa (w przypadku firm) lub nazwisko/imię ( w przypadku osób prywatnych) pasuje do wyszukiwanej frazy.

Publiczna nadpisana metoda ToString(), zwraca napis z wszystkimi transakcjami w bazie danych.

Za pomocą funcki ZapiszXML oraz OdczytXml istnieje możliwość serializacji i deserailizacji danych. Funkcje działają tak samo jak w klasie *Database\_bikes.*

## Obsługa Błędów

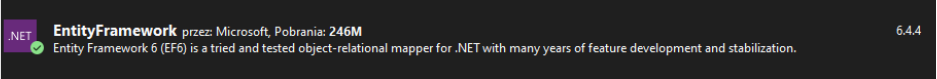
Klasa wyjątków Exceptions zawiera wyjątek InValidDataException dziedziczący po klasie Exception.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

## Obsługa baz danych SQL

System również umożliwia zapisywanie danych do baz danych SQL przy użyciu pakietu Nuget EntityFramework.



Klasa ShopDbContext dziedziczy po DbContext, klasie umożliwiającej zapis do baz SQL. Wewnatrz klasy tworzone sa 3 DbSety odpowiadające tabelą .

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Ponad to w każdej z trzech wymienionych klas istnieje metoda Zapiszdobazydanych , oraz wirtualne metody ustalające połączenia ( np. jeden do wielu) .

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

## Testy jednostkowe

Testy jednostkowe zostały przeprowadzone w osobnym projekcie testów MSTest o nazwie Database\_tests. Projekt jest połączony do projektu systemu.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Za pomocą metody AAA przeprowadzone zostały 2 testy dotyczące dodawania obiektów do klas database oraz usuwania ich.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Poprawnośc dodawania obiektów jest sprawdzna za pomocą weryfikacji czy zwiększyła się ilość obiektów w liście.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Poprawnośc usuwania obiektów jest sprawdzana za pomocą weryfikacji czy zmniejszyła się ilośc obiektów na liście.