Dokumentacja Projektu
Przetwarzanie własnych typów danych CLR UDT
Baza danych pracowników w firmie

Autor

Mikołaj Baczyński
nr. 297846

Spis treści:

1) Opis problemu

Celem projektu było zaimplementowanie aplikacji terminalowej, która obsługuje User Data Types. Aplikacja ma mieć możliwość
wprowadzania danych do zdefiniowanych struktur, wyszukania danych, oraz tworzenie odpowiednich raportów, jak również informować o błędnym ich wykorzystaniu.

2) Opis funkcjonalności udostępnianej przez API

Stworzona aplikacja służy do przechowywania informacji na temat pracowników. Baza danych pozwala na przechowywanie danych osobowych takich jak: adres zamieszkania, pesel, numer konta bankowego. Każdemu pracownikowi możemy przydzielić sprzęt służbowy np. telefon, laptop, samochód. API korzysta z następujących UDT:
- Person
- Location
- Account Number
- Phone
- Laptop
- Car

Korzystanie z tej funkcjonalności umożliwia aplikacja konsolowa. Wybieranie opcji odbywa się przez wciskanie klawiszy, zgodnie z informacjami zawartymi w Menu.

2.1) Menu główne

Możliwe opcje:
- 1 – wyświetlanie danych
- 2 – wprowadzanie danych
- 3 – usuwanie danych
- 0 – wyjście z programu

2.2) Menu wyświetlania

Możemy wyświetlić następujące informacje:
- 1 - Pracownika/ów
- 2 - Sprzęt Pracownika/ów
- 3 – Nr. Telefonu Pracownika/ów
- 4 – Nr. Konta Pracownika/ów
- 5 – Wszystkie Samochody
- 6 – Wszystkie Laptopy
- 7 – Wszystkie Telefony
- 8 – Dostępne Samochody
- 9 – Dostępne Laptopy
- 10 – Dostępne Telefony

W pierwszych czterech opcjach mamy możliwość wyświetlenia wszystkich pracowników, nie podając żadnego argumentu. Lub wyświetlenia jednej osoby podając nazwisko.

2.3) Menu wprowadzania

Wstaw:
- 1 – Pracownika
- 2 – Telefon
- 3 – Laptop
- 4 – Samochód
- 5 – Laptop dla pracownika
- 6 – Telefon dla pracownika
- 7 – Samochód dla pracownika

Opcje 5-7: przydzielenie sprzętu pracownikowi odbywa się przez podanie zarówno id, pracownika jak i sprzętu.

2.4) Menu usuwania:

Usuń:
- 1 – Pracownika
- 2 – Telefonu
- 3 – Laptop
- 4 – Samochód

Usuwanie realizowane za pomocą podania id.

3) Opis typów danych oraz metod (funkcji) udostępnionych w ramach API

3.1) Opis klas User Data Types

Każda klasa ma kilka takich samych metod:

- string ToString() - zwraca informacje o obiekcie
- Parse() - parsuje SqlString, gdy jest on poprawny tworzy nowy obiekt
- Write()/Read() - Serializuje/Deserializuje obiekt
- konstruktory - przyjmują informacje o obiekcie

1. AccountNr – typ reprezentuje numer konta bankowego.

a) Pola klasy:
- string _accountNr – numer konta banku
- string _bankType – typ banku (ING,NBP,itp.)
- enum Bank – identyfikatory poszczególnych banków

b) Metody
- GetAccountNumber() - zwraca numer konta bankowego
- Validate() - zapewnia, by numer banku miał 26 cyfr, oraz sprawdza, czy cyfry 2-6 można dopasować z identyfikatorem banku.

2. Car – typ reprezentuje informacje o samochodzie

a) Pola klasy:
- string _firm – nazwa firmy samochodu
- string _model – model samochodu
- string _plate – numer rejestracyjny

b) Metody
- GetNr() - zwraca numer rejestracyjny
- Validate() - zapewnia by numer rejestracyjny miał 7 znaków z czego pierwsze 2 muszą być dużymi literami.

3. Laptop – typ reprezentuje informacje o laptopie

a) Pola klasy:
- string _firm – nazwa firmy
- string _model – model
- string _serialNumber – numer seryjny

4. Location – typ reprezentuje adres zamieszkania

a) Pola klasy:
- string _city – nazwa miasta
- string _street – nazwa ulicy
- Int32 _nr - numer domu
- Int32 _postCode – kod pocztowy

b) Metody
- Validate() - zapewnia by numer domu był liczbą większą od zera, oraz kod pocztowy miał 5 cyfr.

5. Person – typ reprezentuje dane osobowe

a) Pola klasy:
- string _surname – nazwisko
- string _name – imię
- string _sex - płeć
- Int64 _pesel - pesel

b) Metody
- ValidateSex() - zapewnia by płeć miała wartości kobieta/meszczyzna. Nie bierze pod uwagę wielkości znaków.
- ValidatePesel() - zapewnia by pesel miał 11 cyfr, oraz sprawdza czy 9-cyfra jest parzysta dla mężczyzny i nieparzysta dla kobiety.
- GetSurname() - zwraca nazwisko
- GetPesel() - zwraca pesel

6. Car – typ reprezentuje informacje o telefonie

a) Pola klasy:
- string _firm – nazwa firmy telefonu
- string _model – model telefonu
- string _nrPhone – numer telefonu

b) Metody
- GetPhoneNumber() - zwraca numer telefonu
- Validate() - zapewnia by numer telefonu miał 9 cyfr

3.2) Opis Tabel wykorzystanych w bazie danych

Tabele znajdują się w bazie danych FIRMA.

1. pracownik:
- id INT PK
- osoba dbo.Person NOT NULL,
- adres dbo.Location NOT NULL,
- nrKonta dbo.AccountNr NOT NULL,

2. laptop:
- id INT PK
- laptop dbo.Laptop NOT NULL

3. telefon:
- id INT PK
- telefon dbo.Phone NOT NULL,

4. samochod:
- id int PRIMARY KEY IDENTITY (1, 1),
- telefon dbo.Car NOT NULL,

5. prac_sprzet(asocjacyjna):
- id_pracownik int UNIQUE NOT NULL,
- id_laptop int FK,
- id_telefon int FK,
- id_samochod int FK,

3.3) Opis Klas Aplikacji Konsolowej

Aplikacja składa się z następujących klas:

1. Select - Klasa reprezentująca zapytanie Select

a) Pola klasy:
- string ConnOption – dane dotyczące połączenia z bazą danych
- List<string> command – zawiera wykorzystywane zapytania

b) Metody:
- ExecuteSelect(int nr, string Surname) – łączy się z bazą danych. Wykonuje zapytanie znajdujące się pod indeksem nr – 1 w liście command. Do zapytania jest również dołączana zmienna Surname, gdy chcemy wyświetlić konkretną osobę.

2. Insert - Klasa reprezentująca polecenie Insert

a) Pola klasy:
- string ConnOption – dane dotyczące połączenia z bazą danych
- List<string> command – zawiera wykorzystywane polecenia

b) Metody:

- ExecuteInsert(int nr, string data) – łączy się z bazą danych. Wykonuje polecenie znajdujące się pod indeksem nr – 1 w liście command. Do polecenia jest również dołączana zmienna data, zawierająca dane do wstawienia.
- ExecuteUpdate(int nr, string WorkerID, string EquipmentID) - realizuje funkcjonalność przypisania/odebrania sprzętu do pracownika. Wykonuje polecenie UPDATE, gdy pracownik posiada jakikolwiek sprzęt, lub wstawia, gdy dostaje po raz pierwszy.
- PeselExists(string pesel) – sprawdza, czy w bazie istnieje pracownik danym peselem. Pozwala zapobiec dodania osób z takim samym peselm.
- PhoneNrExists(string nr) - sprawdza, czy w bazie istnieje telefon o danym numerze. Pozwala zapobiec dodania telefonu z takim samym numerem telefonu.
- WorkerExistsInEquipment(string WorkerID) – sprawdza, czy pracownik istnieje w tabeli prac_sprzet. Metoda decyduje, czy w ExecuteUpdate() będzie wykonywane polecenie Update, czy Insert.

3. Delete - Klasa reprezentująca polecenie Delete

a) Pola klasy:

- string ConnOption – dane dotyczące połączenia z bazą danych
- List<string> command – zawiera wykorzystywane polecenia

b) Metody:

- ExecuteDelete(int nr, string id) – łączy się z bazą danych. Wykonuje polecenie znajdujące się pod indeksem nr – 1 w liście command. Do polecenia jest również dołączana zmienna id, by usunąć element o konkretnym id.

4. Menu - Klasa reprezentująca wyświetlenie i obsługę poszczególnych Menu.

a) Pola klasy:

- Brak

b) Metody wypisujące dane Menu:

- PrintMenu()
- PrintSelectMenu()
- PrintInsertMenu()
- PrintDeleteMenu()

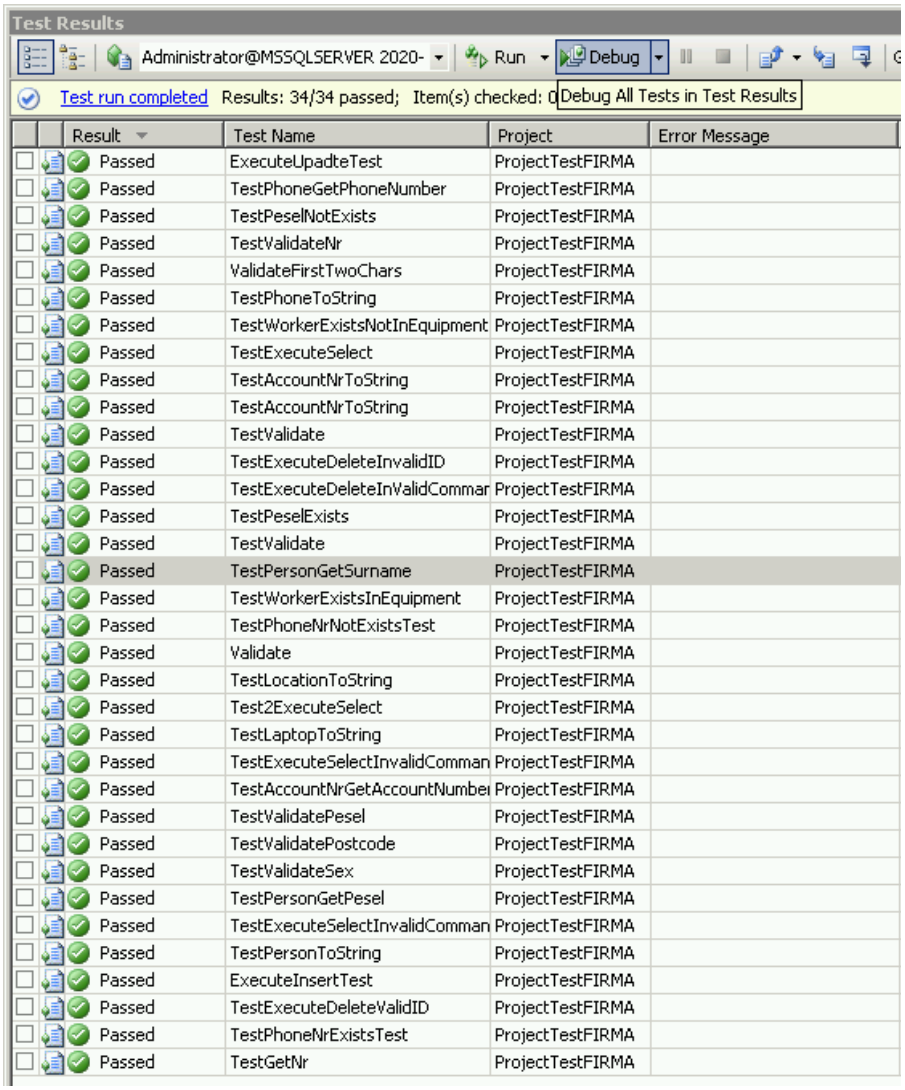
c) Metody obsługujące konkretne Menu:

- HandleMenu()
- HandleMenu()
- HandleMenu()
- HandleMenu()

5. Program - Jest to główna klasa programu. W funkcji Main wywoływane są metody klasy Menu.

4.) Prezentacja przeprowadzonych testów jednostkowych.

W projekcie zostało przeprowadzonych 34 testów jednostkowych. Została przetestowana, każda metoda klas UDT, oraz metody klas Programu: Insert, Select, Delete. Metody klasy Menu nie zostały przetestowane, ponieważ mają na celu wypisanie menu na ekran, oraz pobranie znaków. Poniżej znajduje się wykaz przeprowadzonych testów.



Result	Test Name	Project	Error Message
Passed	ExecuteUpadteTest	ProjectTestFIRMA	
Passed	TestPhoneGetPhoneNumber	ProjectTestFIRMA	
Passed	TestPeselNotExists	ProjectTestFIRMA	
Passed	TestValidateNr	ProjectTestFIRMA	
Passed	ValidateFirstTwoChars	ProjectTestFIRMA	
Passed	TestPhoneToString	ProjectTestFIRMA	
Passed	TestWorkerExistsNotInEquipment	ProjectTestFIRMA	
Passed	TestExecuteSelect	ProjectTestFIRMA	
Passed	TestAccountNrToString	ProjectTestFIRMA	
Passed	TestAccountNrToString	ProjectTestFIRMA	
Passed	TestValidate	ProjectTestFIRMA	
Passed	TestExecuteDeleteInvalidID	ProjectTestFIRMA	
Passed	TestExecuteDeleteInValidCommar	ProjectTestFIRMA	
Passed	TestPeselExists	ProjectTestFIRMA	
Passed	TestValidate	ProjectTestFIRMA	
Passed	TestPersonGetSurname	ProjectTestFIRMA	
Passed	TestWorkerExistsInEquipment	ProjectTestFIRMA	
Passed	TestPhoneNrNotExistsTest	ProjectTestFIRMA	
Passed	Validate	ProjectTestFIRMA	
Passed	TestLocationToString	ProjectTestFIRMA	
Passed	Test2ExecuteSelect	ProjectTestFIRMA	
Passed	TestLaptopToString	ProjectTestFIRMA	
Passed	TestExecuteSelectInvalidComman	ProjectTestFIRMA	
Passed	TestAccountNrGetAccountNumbe	ProjectTestFIRMA	
Passed	TestValidatePesel	ProjectTestFIRMA	
Passed	TestValidatePostcode	ProjectTestFIRMA	
Passed	TestValidateSex	ProjectTestFIRMA	
Passed	TestPersonGetPesel	ProjectTestFIRMA	
Passed	TestExecuteSelectInvalidComman	ProjectTestFIRMA	
Passed	TestPersonToString	ProjectTestFIRMA	
Passed	ExecuteInsertTest	ProjectTestFIRMA	
Passed	TestExecuteDeleteValidID	ProjectTestFIRMA	
Passed	TestPhoneNrExistsTest	ProjectTestFIRMA	
Passed	TestGetNr	ProjectTestFIRMA	

1) Zrzut wyników testów

5.) Wnioski

Dzięki możliwości tworzenia User Data Types można w bardzo łatwy sposób, używając języka C#, tworzyć własne typy danych. W moim projekcie większość typów można by było przechowywać korzystając ze zwykłych typów. Na przykład stworzony przeze mnie typ Person, reprezentujący dane osobowe, mógłby zostać zrealizowany jako tabela z atrybutami odpowiadającymi polom klasy Person.

6.) Literatura

- https://newton.fis.agh.edu.pl/~antek/
- https://docs.microsoft.com/pl-pl/dotnet/api
- https://stackoverflow.com/

7.) Kody Źródłowe

a) Skrypty T-SQL – znajdują się w folderze /*ProjektMikolajBaczynski*/SQLConfig:
- create.sql – tworzy bazę danych, oraz tabele.
- delete.sql – usuwa bazę danych, oraz tabele.
- insert.sql – wstawienie do bazy przykładowych danych

b) Pliki klas UDT – znajdują się w folderze /*ProjektMikolajBaczynski*/UDT:
- Car.cs – implementacja typu UDT – car
- Laptop.cs - implementacja typu UDT – laptop
- Person.cs - implementacja typu UDT – person
- Phone.cs - implementacja typu UDT – phone
- AccountNr.cs - implementacja typu UDT – accountNr
- Location.cs - implementacja typu UDT – location

c) Pliki Programu - znajdują się w folderze /*ProjektMikolajBaczynski*/Program:
- Program.cs – kod z główną pętlą programu
- Menu.cs – kod z klasą obsługującą menu
- Select.cs – kod z klasą wykonywającą zapytania Select
- Insert.cs - kod z klasą wykonywającą polecenia Insert
- Delete.cs - kod z klasą wykonywającą polecenia Delete

d) Pliki z testami - znajdują się w folderze /*ProjektMikolajBaczynski*/Tests:
pliki nazywają się tak samo jak wymienione pliki w podpunktach b) i c) z taką różnicą, że nazwy dodane jest słowo „Test”

8.) Uruchomienie:

Aplikacja:
- Przechodzimy do folderu /*ProjektMikolajBaczynski/ProjectAppFirma*
- Otwieramy projekt *ProjectAppFirma.sln* za pomocą *VisualStudio*
- Po lewej stronie w okienku *SolutionExplorer* kilkamy prawym przyciskiem myszy na projekt ProjektFIRMAv2 → *Deploy*
- Następnie uruchamiamy skrypt */SQLConfig/create.sql* oraz *insert.sql*
- Uruchamiamy aplikacje *ProjectAppFirma.exe*

Testy:
- otwieramy projekt *ProjectAppFirma.sln*
- Po lewej stronie w okienku *SolutionExplorer* ,wybieramy z projektu *AppTest* dowolną klasę z testami
- prawym przyciskiem myszy klikamy na kod i następnie *Run Tests*