Obsługa danych przestrzennych dwuwymiarowych

Bazy danych II – dokumentacja projektu

Michał Orlewski

Repozytorium: https://github.com/m-orlewski/2d-spatial-data-processing-api

1. Opis problemu i opis funkcjonalności udostępnianej przez API

Zadanie polegało na przygotowaniu API i jego implementacji umożliwiającej przetwarzanie własnych typów danych CLR UDT oraz metod. Opracowane typy danych i metody porównano z typami przetwarzającymi dane przestrzenne udostępnionymi w ramach SQL Server.

Funkcjonalności udostępnione przez API umożliwiają dodawanie, wyświetlanie i usuwanie obiektów utworzonych typów danych z bazy danych oraz uruchamianie ich metod.

2. Opis typów danych oraz metod udostępnionych w ramach API

W ramach projektu utworzono 4 typy danych reprezentujące obiekty dwuwymiarowe:

1) Point
    - *double x* – współrzędna x punktu
    - *double y* – współrzędna y punktu
    - *bool isNull*

    Typ Point reprezentuje punkt w przestrzeni dwuwymiarowej, w ramach klasy udostępniono następujące metody:

    - *bool IsNull()* – zwraca true jeżeli obiekt jest null
    - *static Point Null* – zwraca obiekt o wartości null
    - *double X* – getter i setter pola x
    - *double Y* – getter I setter pola y
    - *string ToString()* – konwertuje obiekt do typu string
    - *static Point Parse(SqlString)* – parser zwracający obiekt z typu SqlString
    - *double DistanceFrom(Point)* – zwraca odległość od punktu
    - *bool IsInsideCircle(Circle)* – zwraca true jeżeli punkt leży wewnątrz okręgu
    - *bool IsInsideTriangle(Triangle)* – zwraca true jeżeli punkt leży wewnątrz trójkąta
    - *bool IsInsideQuadrangle(Quadrangle)* – zwraca true jeżeli punkt leży wewnątrz czworokąta
2) Circle
    - *Point c* – punkt reprezentujący środek okręgu
    - *double r* – promień okręgu
    - *bool isNull*

Typ Circle reprezentuje okrąg w przestrzeni dwuwymiarowej, w ramach klasy udostępniono następujące metody:

- *bool IsNull()* – zwraca true jeżeli obiekt jest null
- *static Circle Null* – zwraca obiekt o wartości null
- *Point C* – getter i setter pola c
- *double R* – getter I setter pola r
- *string ToString()* – konwertuje obiekt do typu string
- *static Circle Parse(SqlString)* – parser zwracający obiekt z typu SqlString
- *bool ValidateCircle()* – zwraca true jeżeli obiekt jest poprawny
- *double getSurfaceArea()* – zwraca pole obiektu Circle

3) Triangle
    - *Point p1, p2, p3* – punkty reprezentujące wierzchołki trójkąta
    - *bool isNull*

   Typ Triangle reprezentuje trójkąt w przestrzeni dwuwymiarowej, w ramach klasy udostępniono następujące metody:

    - *bool IsNull()* – zwraca true jeżeli obiekt jest null
    - *static Triangle Null* – zwraca obiekt o wartości null
    - *Point P1, P2, P3* – gettery i settery pól p1, p2, p3
    - *string ToString()* – konwertuje obiekt do typu string
    - *static Triangle Parse(SqlString)* – parser zwracający obiekt z typu SqlString
    - *bool ValidateTriangle()* – zwraca true jeżeli obiekt jest poprawny
    - *double getSurfaceArea()* – zwraca pole obiektu Triangle

4) Quadrangle
    - *Point p1, p2, p3, p4* – punkty reprezentujące wierzchołki czworokąta
    - *bool isNull*

   Typ Quadrangle reprezentuje czworokąt w przestrzeni dwuwymiarowej, w ramach klasy udostępniono następujące metody:

    - *bool IsNull()* – zwraca true jeżeli obiekt jest null
    - *static Quadrangle Null* – zwraca obiekt o wartości null
    - *Point P1, P2, P3* – gettery i settery pól p1, p2, p3
    - *string ToString()* – konwertuje obiekt do typu string
    - *static Quadrangle Parse(SqlString)* – parser zwracający obiekt z typu SqlString
    - *bool ValidateQuadrangle()* – zwraca true jeżeli obiekt jest poprawny
    - *double getSurfaceArea()* – zwraca pole obiektu Quadrangle

W ramach API udostępniono następujące możliwości:

1. Wyświetlanie danych z bazy
    a. Wyświetlenie wszystkich punktów z bazy
    b. Wyświetlenie wszystkich okręgów z bazy
    c. Wyświetlenie wszystkich trójkątów z bazy
    d. Wyświetlenie wszystkich czworokątów z bazy
2. Dodawanie danych do bazy
    a. Dodanie punktu do bazy
    b. Dodanie okręgu do bazy
    c. Dodanie trójkąta do bazy
    d. Dodanie czworokąta do bazy
3. Wykonywanie operacji na danych z bazy
    a. Obliczanie odległości między punktami
    b. Obliczanie pola okręgu
    c. Obliczanie pola trójkąta
    d. Obliczanie pola czworokąta
    e. Sprawdzanie czy punkt jest wewnątrz okręgu
    f. Sprawdzanie czy punkt jest wewnątrz trójkąta
    g. Sprawdzanie czy punkt jest wewnątrz czworokąta
4. Usuwanie danych z bazy
    a. Usuwanie punktu z bazy
    b. Usuwanie okręgu z bazy
    c. Usuwanie trójkąta z bazy
    d. Usuwanie czworokąta z bazy

3. Opis implementacji

Projekt wykonano w Visual Studio 2022 przy użyciu następujących technologii:

- C#
- .NET 4.7.2
- SQL Server 2016

Rozwiązanie składało się z 3 projektów:

- Aplikacja konsolowa .NET – implementacja API
- SQL Server Database Project – implementacja CLR UDT
- Projekt testów jednostkowych (.NET Framework) – testy utworzonych UDT

Połączenie z lokalną bazą danych wykonano przy użyciu interfejsu ADO.NET:

```
SqlConnection conn = null;

// test połączenia
try
{
    string connString = "Persist Security Info=False;Trusted_Connection=True;database=DB2_project;server=(local)"; // wymaga zmian do uruchomienia lokalnego
    conn = new SqlConnection(connString);
    conn.Open();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    conn.Close();
}
```

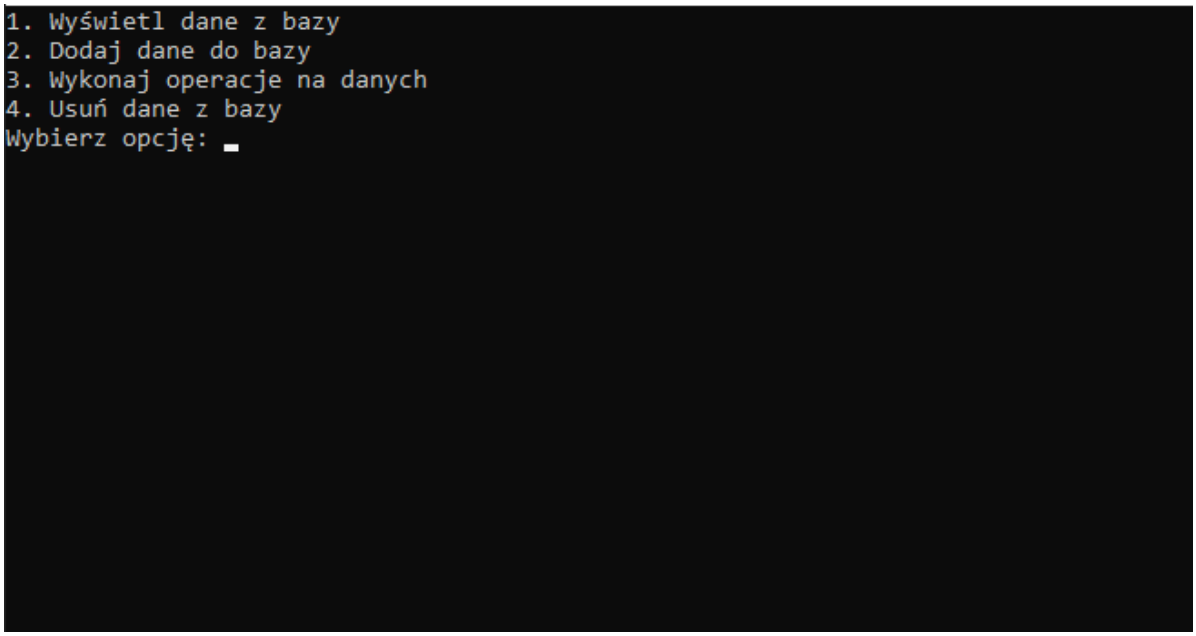
Rys. 1: Połączenie ADO.NET z bazą danych

Implementacji API znajduje się w 5 klasach:

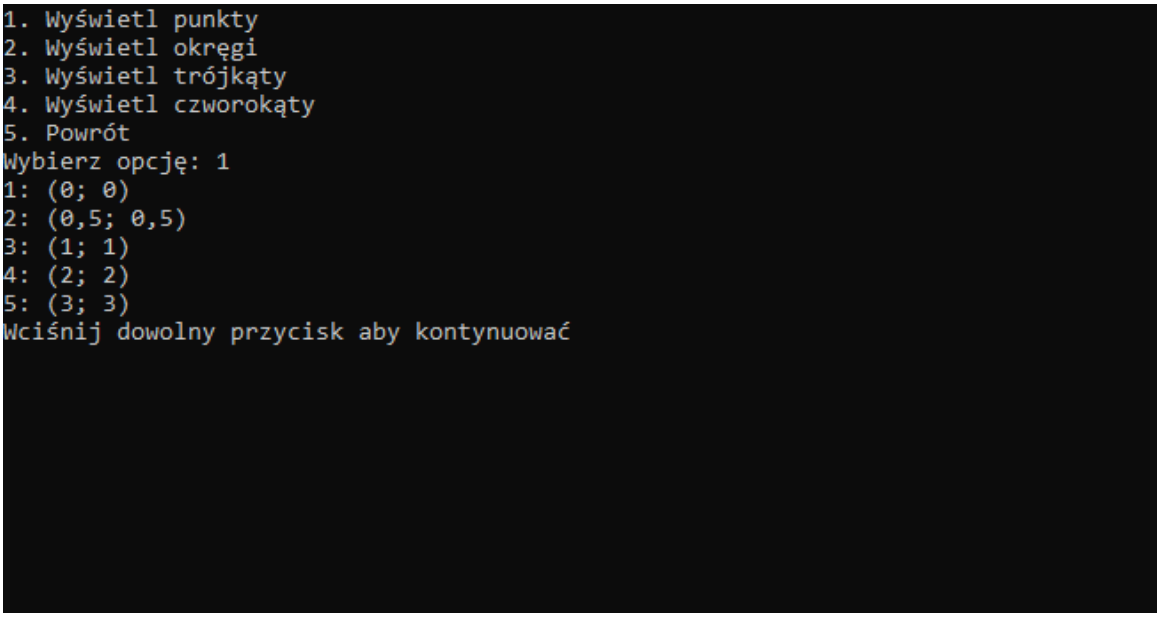
- Program – główna klasa sterująca API
- Selector – klasa odpowiedzialna za wyświetlanie danych z bazy
- Inserter – klasa odpowiedzialna za dodawanie danych do bazy
- Calculator – klasa odpowiedzialna za wykonywanie operacji na danych
- Deleter – klasa odpowiedzialna za usuwanie danych z bazy

Poszczególne klasy pobierają dane wejściowe od użytkownika, tworzą i wysyłają zapytania SQL do bazy danych oraz wyświetlają otrzymane wyniki.

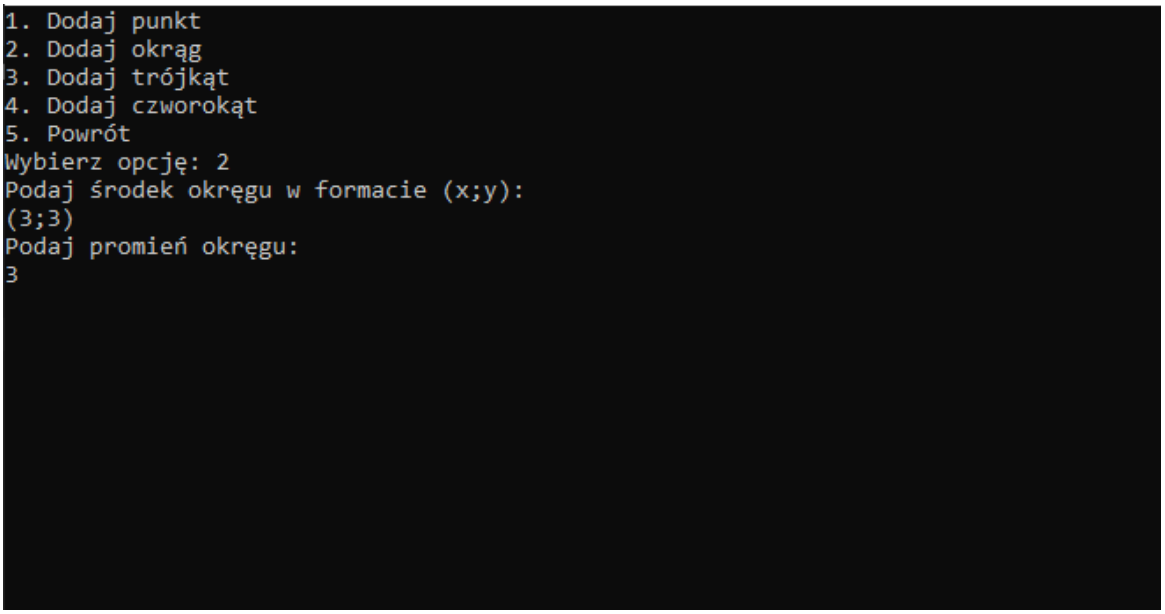
Prezentacja działania API:



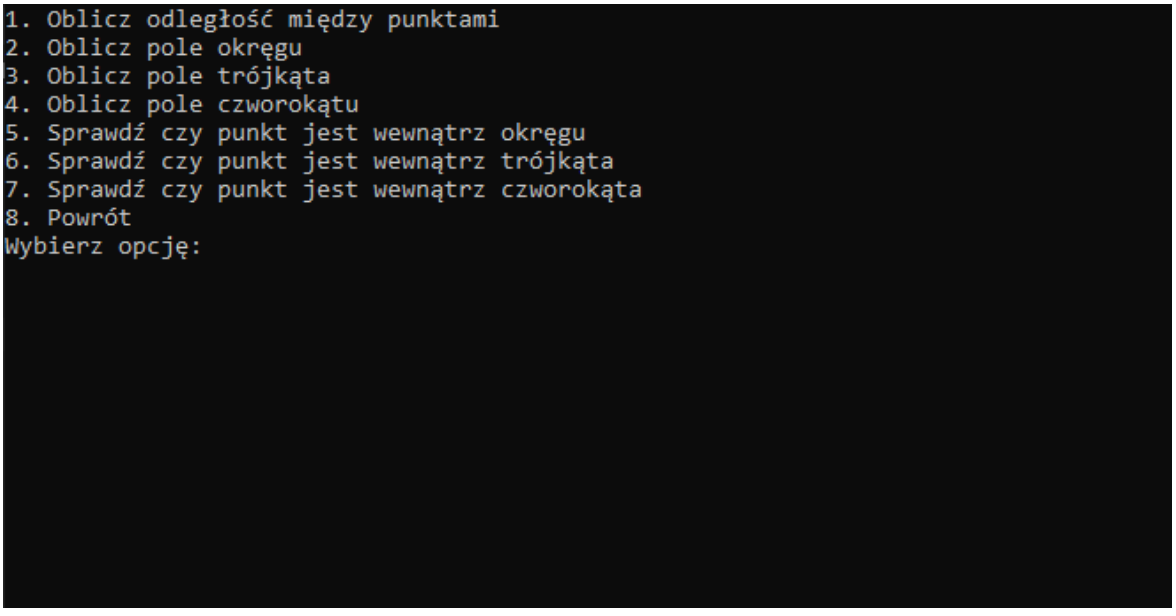
Rys. 2: Główny ekran API



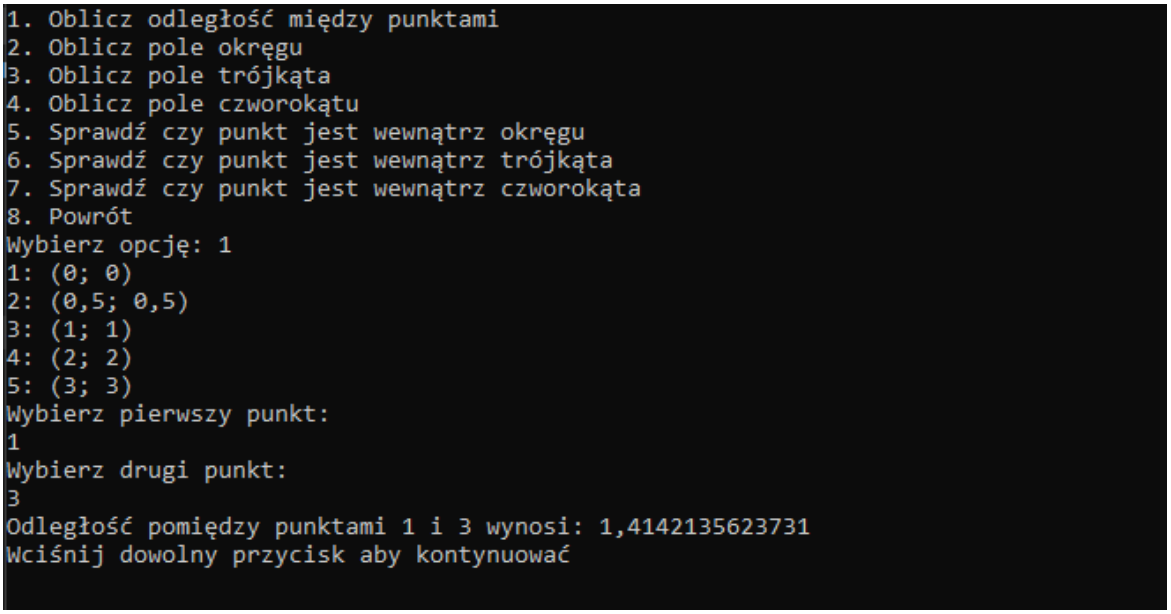
Rys. 3: Wyświetlanie danych z bazy



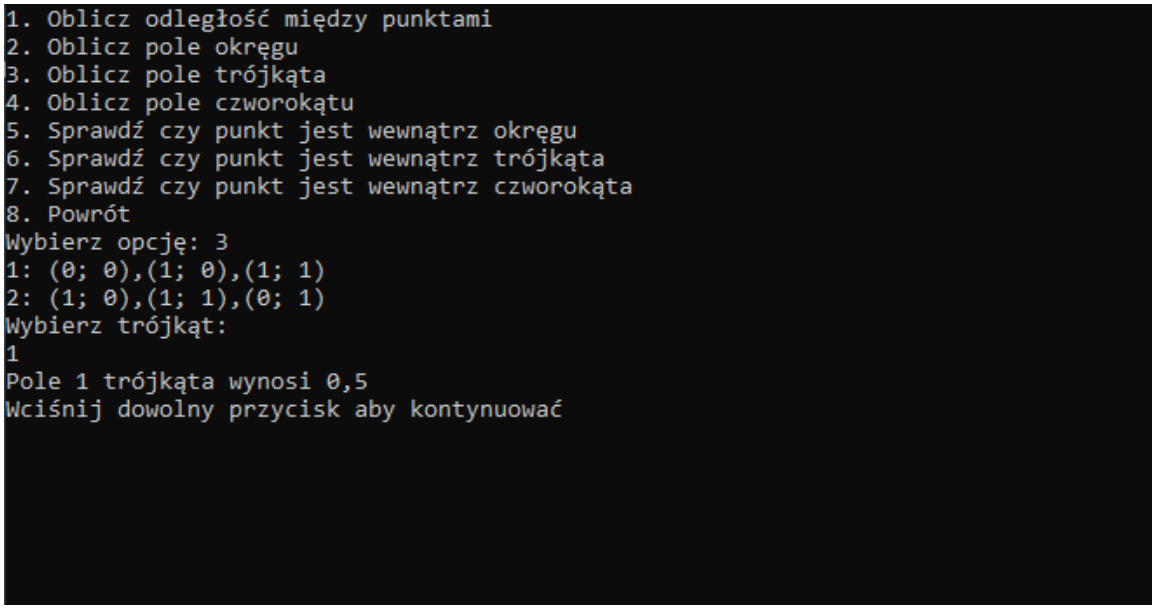
Rys. 4: Dodawanie danych do bazy



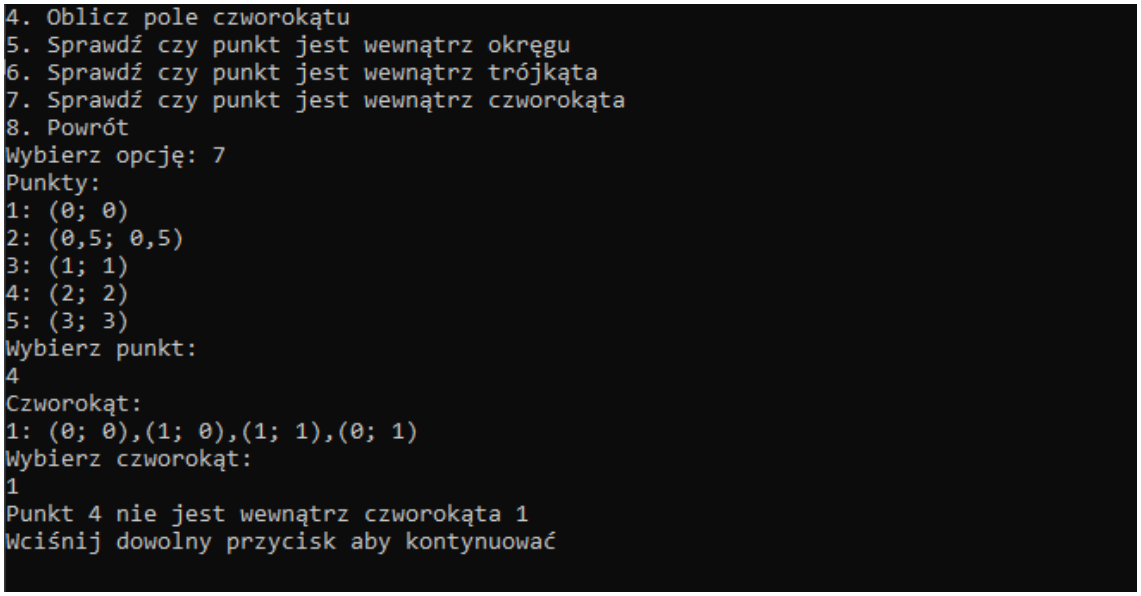
Rys. 5: Ekran API do wykonywania operacji na danych



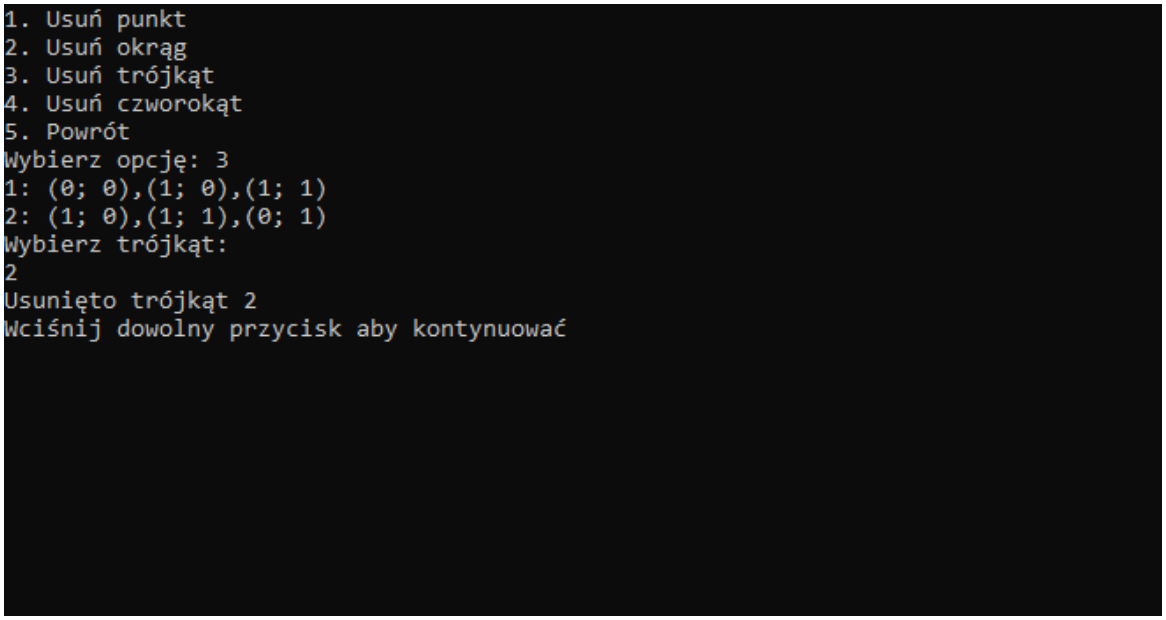
Rys. 6: Obliczenie odległości między punktami



Rys. 7: Obliczenie pola trójkąta

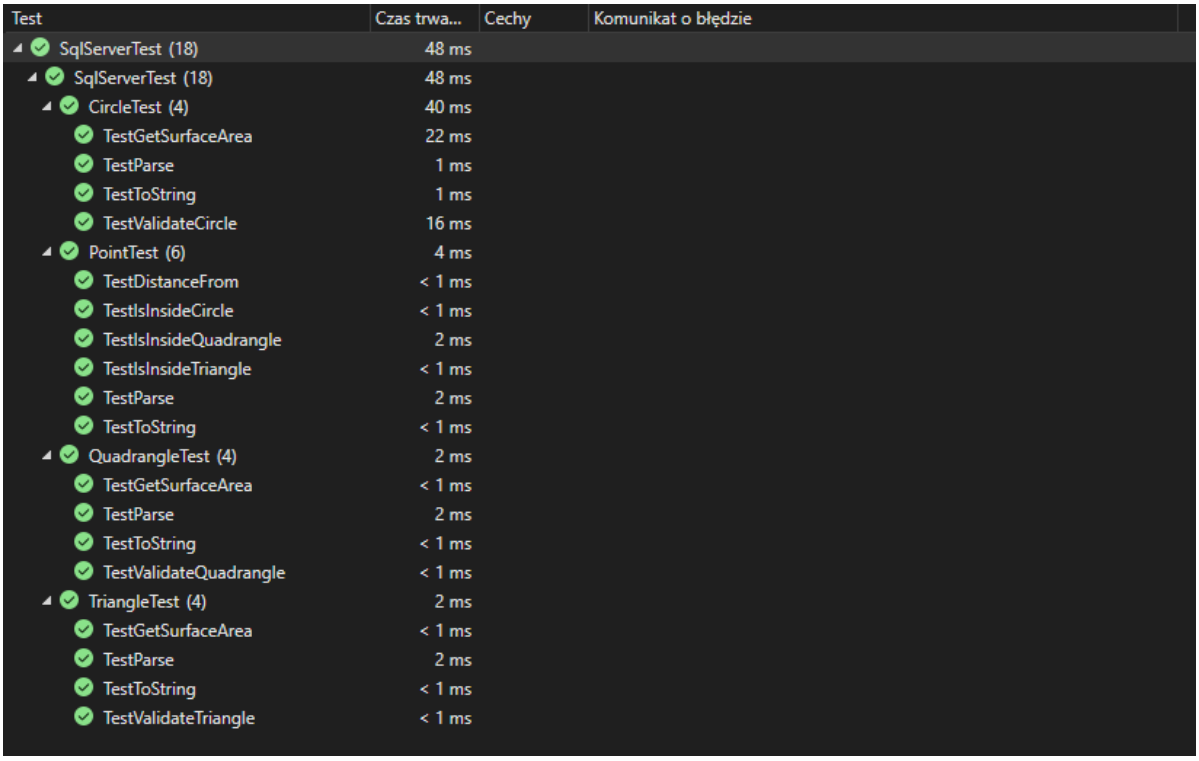


Rys. 8: Sprawdzenie czy punkt leży wewnątrz czworokąta



Rys. 9: Usuwanie danych z bazy

Poprawne działanie utworzonych typów danych zostało zweryfikowane testami jednostkowymi. Przetestowano poszczególne metody, biorąc pod uwagę różne przypadki.



Rys. 10: Prezentacja wyników przeprowadzonych testów jednostkowych

W celu testowania aplikacji utworzono skrypt *SQL/InitializeDbScript.sql,* który tworzy w bazie danych tabele przechowywujące utworzone typy danych oraz wypełnia je przykładowymi danymi.

4. Porównanie utworzonych typów danych i metod z typami udostępnionymi w ramach SQL Server

W celu porównania utworzono skrypt *SQL/CompareWithGeometryScript.sql.*

W skrypcie porównano utworzone typy danych z wbudowanym typem *geometry*. Porównano składnię, możliwości oraz metody typów.

	myPoint	sqlPoint	myDistance	sqlDistance
1	(1; 1)	POINT (1 1)	0	0
2	(2; 2)	POINT (2 2)	1,4142135623731	1,4142135623731

Rys. 11: Porównanie punktów i funkcji wyliczającej odległość (od punktu 1)

	myCircle	sqlCircle	myIsInsideCircle	sqlIsInsideCircle	myCircleArea	sqlCircleArea
1	c=(0; 0) r=2	CIRCULARSTRING (1 0, 0 1, -1 0, 0 -1, 1 0)	1	0	12,5663706143592	0

Rys. 12: Porównanie okręgów, metod sprawdzających czy punkt 1 należy do okręgu oraz obliczających pole okręgu

	myTriangle	sqlTriangle	myIsInsideTriangle	sqlIsInsideTriangle	myTriangleArea	sqlTriangleArea
1	(0; 0),(1; 0),(0; 1)	POLYGON ((0 0, 1 0, 0 1, 0 0))	0	0	0,5	0,5

Rys. 13: Porównanie trójkątów, metod sprawdzających czy punkt 1 należy do trójkąta oraz obliczających pole trójkąta

	myQuadrangle	sqlQuadrangle	myIsInsideQuadrangle	sqlIsInsideQuadrangle	myQuadrangleArea	sqlQuadrangleArea
1	(0; 0),(1; 0),(1; 1),(0; 1)	POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))	1	0	1	1

Rys. 14: Porównanie czworokątów, metod sprawdzających czy punkt 1 należy do czworokątu oraz obliczających pole czworokąta

5. Wnioski i podsumowanie

Utworzony system skutecznie implementuje założone funkcjonalności(*rys. 2 - 9*), pozwala na tworzenie zaawansowanych typów danych stosując wygodny i rozbudowany język programowania.

Typy danych utworzone w CLR mogą być łatwo wdrożone do bazy danych i stosowane w niej. Rozbudowany framework do pisania testów jednostkowych pozwala na sprawdzenie poprawności kodu w każdym etapie pisania go (*rys. 10*).

Porównując utworzone typy danych z już istniejącymi, wbudowanymi możemy zauważyć parę różnic implementacyjnych. Największą zaletą UDT jest możliwość dostosowania typów danych do własnych potrzeb, poprzez swobodę implementacji oraz dowolność przy tworzeniu metod.

Rys. 12 – 14 pokazują pewne wady wbudowanych typów danych w SQL Server. Typ *geometry::CIRCULARSTRING* nie pozwala w pełni odwzorować okręgu (metody wyliczające pole lub sprawdzające czy punkt zawiera się w okręgu nie działają poprawnie). Można także zauważyć, że typ *geometry::POLYGON* nie uznaje punktów leżących na jednej z krawędzi za punkty należące do jego powierzchni.

Za zaletę typów wbudowanych, a szczególnie typu *geometry::POLYGON,* można uznać jego większą uniwersalność - reprezentuje wielokąt o dowolnej liczbie wierzchołków w przeciwieństwie do utworzonych w projekcie, wyspecjalizowanych, typów danych: Triangle i Quadrangle.

6. Literatura

- https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16
- https://stackoverflow.com/questions/tagged/tsql