Przemysłowe Bazy Danych

Aplikacja testująca szybkość zapisu do plików tekstowych oraz bazy danych MS SQL Server

Autorzy:
Jakub Pająk
Kacper Stiborski

AiR Grupa 5TI

Gliwice 2024

Spis treści

1. Wprowadzenie

1.1. Cel projektu

Celem projektu była implementacja aplikacji desktopowej, której zadaniem było wykonywanie operacji na plikach tekstowych oraz bazie danych. Aplikacja miała umożliwiać zapis określonej przez użytkownika liczby rekordów do pliku tekstowego oraz ich odczyt, a także dokonywać pomiaru czasu niezbędnego do wykonania tych operacji. Dodatkowo, projekt zakładał realizację analogicznych operacji na bazie danych MS SQL Server, z uwzględnieniem zapisu i odczytu rekordów oraz pomiaru wydajności obu procesów. Celem porównania efektywności operacji na plikach oraz bazie danych, aplikacja miała dostarczyć użytkownikowi czytelne informacje o czasie trwania poszczególnych zadań.

2. Realizacja projektu

2.1. Interfejs użytkownika

Interfejs użytkownika został zaprojektowany przy użyciu wbudowanego projektanta graficznego, będącego częścią frameworka *Windows Forms*. Proces ten polegał na wybieraniu z przybornika (ang. *Toolbox*) odpowiednich elementów funkcjonalnych, takich jak przyciski, etykiety czy pola tekstowe, i umieszczaniu ich na obszarze roboczym okna aplikacji. Dzięki narzędziu *Properties*, możliwe było szczegółowe dostosowanie wyglądu i zachowania poszczególnych kontrolek do specyficznych wymagań projektowych, takich jak rozmiar, położenie, czcionka, czy reakcje na zdarzenia użytkownika.

2.2. Obsługa zdarzeń i logika aplikacji

Funkcja odpowiedzialna za inicjalizację oraz obsługę zdarzeń na głównym widoku aplikacji ma prostą, lecz efektywną strukturę. W konstruktorze następuje inicjalizacja widoku oraz przypisanie wartości początkowych zmiennym niezbędnym do działania aplikacji. Wszystkie kolejne funkcje związane z obsługą zdarzeń są generowane automatycznie przez Visual Studio, na przykład poprzez dwukrotne kliknięcie na kontrolkę w widoku projektanta. Tak właśnie zrealizowano obsługę przycisków w opisywanej aplikacji.

Główna funkcja logiki aplikacji wywołuje dwie metody pomocnicze: *InvokeWriteServices()* oraz *InvokeReadServices()*. Metody te z kolei korzystają z odpowiednich klas, które obsługują zapis i odczyt danych, zwracając wyniki w postaci zmiennej typu *Dictionary<string, double>*. Klucz typu *string* odpowiada za identyfikację typu pliku, natomiast wartość typu *double* przechowuje czas potrzebny na wykonanie operacji.

Następnie, czasy operacji są przypisywane do obiektów typu *Label*, co pozwala na ich wyświetlenie w interfejsie użytkownika, dostarczając czytelne informacje o wydajności działania aplikacji.

2.3. Implementacja metod zapisu do pliku

Wszystkie metody odpowiedzialne za zapis do pliku tekstowego są do siebie bardzo podobne. Pierwszym krokiem jest wybranie odpowiedniej ścieżki, w której plik zostanie zapisany. W języku C# można wykorzystać klasę *Environment*, co umożliwia tworzenie dynamicznych ścieżek zależnych od systemu operacyjnego użytkownika.

Kolejny etap to wygenerowanie odpowiedniej ilości rekordów. Dane są generowane za pomocą biblioteki *Bogus*, która pozwala na generowanie realistycznie wyglądających danych testowych, co znacząco ułatwia testowanie aplikacji.

Zasadnicza różnica między klasami obsługującymi zapis do plików różnych typów dotyczy jedynie metody samego zapisu. Dla plików binarnych używana jest klasa *BinaryWriter*, natomiast dla plików tekstowych i CSV stosuje się klasę *StreamWriter*. Dzięki temu każda metoda zapisu jest dostosowana do specyfiki odpowiedniego formatu pliku.

2.4. Implementacja zapisu do bazy danych

Proces zapisu do bazy danych wymagał zainstalowania odpowiednich paczek, aby umożliwić korzystanie z *Entity Framework*. Wymagane paczki to:

- *Microsoft.EntityFrameworkCore*
- *Microsoft.EntityFrameworkCore.SqlServer*
- *Microsoft.EntityFrameworkCore.Tools*

Dodatkowo, do przeprowadzenia migracji, czyli sposobu, w jaki *Entity Framework* zarządza zmianami w strukturze bazy danych, konieczne było upewnienie się, że pakiet *.NET CLI* jest zainstalowany.

Pierwszym krokiem integracji aplikacji z bazą danych było stworzenie modelu danych. W omawianej aplikacji model ten jest reprezentowany przez klasę *BasicDataDto.cs*, która odpowiada encji bazodanowej. Następnie, w celu skonfigurowania modelu, konieczne było nadpisanie metody *OnModelCreating()* w klasie dziedziczącej po *DbContext*. Taki zabieg informuje *Entity Framework*, że dana klasa odpowiada za konfigurację struktury bazy danych.

Aby aplikacja mogła komunikować się z bazą danych, musiała zostać uruchomiona wewnątrz klauzuli *using* w funkcji *Main()*, co zapewnia odpowiednie zarządzanie połączeniami z bazą podczas działania programu.

Sam proces zapisu danych do bazy jest stosunkowo prosty. Do kontekstu bazy danych dodawane są nowe rekordy, po czym metoda *SaveChanges()* zapisuje zmiany w bazie danych, zapewniając trwałość operacji.