Laboratorium 3 Sprawozdanie z realizacji laboratorium			
Temat: Praktyczne wykorzystanie ORM	**Nr Albumu:** 028487	**Grupa/zespół:** GL01	**Rok/semestr:** III / 6
Wykonał: Oleksii Hudzishevskyi	**Data wykonania:** 24/04/2023		**Data oddania:** 01/07/2023
	Ocena:		**Podpis prowadzącego:**

1. Spis treści

2. Cel ćwiczenia

Laboratorium ma na celu poszerzenie wiedzy praktycznej w zakresie wykorzystywania narzędzi ORM w trakcie tworzenia aplikacji.

3. Wymagania znajomości zagadnień

- Pisanie prostych aplikacji w C# lub innym obiektowym języku wysokiego poziomu
- Umiejętność tworzenia prostych aplikacji z wykorzystaniem GUI np. WinForms
- Znajomość zagadnień z Laboratorium 2
- Umiejętność pracy na kolekcjach obiektów, proste zapytania LINQ.
- Wskazana podstawowa znajomość języka angielskiego lub też umiejętność korzystania z narzędzi tłumaczenia on-line. Wynika to z faktu, że większość użytecznej i najbardziej aktualnej dokumentacji jest publikowana właśnie w języku angielskim.

4. Literatura, materiały dydaktyczne

- https://www.entityframeworktutorial.net/EntityFramework6/introduction.aspx
- https://www.mdatelier.pl/entity-framework-database-first/
- https://fildev.net/2016/07/28/lazy-loading-vs-eager-loading-entity-framework/
- https://pl.blog.iwanek.eu/lazy-loading-i-eager-loading/
- https://www.sqlpedia.pl/
- https://www.plukasiewicz.net/EFCore/Introduction
- https://www.plukasiewicz.net/EFCore/EFCoreMigration
- https://docs.microsoft.com/pl-pl/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli
- https://www.youtube.com/watch?v=qkJ9keBmQWo
- https://devstyle.pl/2018/09/24/orm-vs-sql/

5. Wiadomości teoretyczne.

W celu przygotowania się do realizacji części praktycznej laboratoriów związanych z wykorzystaniem narzędzi ORM, należy w pierwszej kolejności solidnie opracować zagadnienia

teoretyczne. Głównym zadaniem obecnych laboratoriów jest właśnie przygotowanie takowej bazy wiedzy. Należy odnaleźć w źródłach internetowych oraz ewentualnej literaturze definicję wybranych zagadnień. Uwaga, przez wzgląd na przydatność wymienionych, należy postarać się udzielić obszernych i wyczerpujących odpowiedzi, wraz z ewentualnymi przykładami, nie należy się ograniczać do krótkich jedno lub dwuzdaniowych odpowiedzi.

- Rozwinąć anagram ACID, wypisać oraz opisać znaczenie słów składowych anagramu w kontekście przetwarzania transakcyjnego
- Omówić typowe anomalie transakcji jakie mogą wystąpić
    - Brudny odczyt (Dirty read)
    - Utracona modyfikacja (Lost update)
    - Niepowtarzalny odczyt (Non-repeatable read)
    - Fantomy (Phantoms)
- Omówić 4 poziomy izolacji według standardu SQL-92
    - Niezatwierdzony Odczyt (READ UNCOMMITTED)
    - Odczyt Zatwierdzonych Danych (READ COMMITTED)
    - Powtarzalny Odczyt (REPEATABLE READ)
    - Uszeregowany (SERIALIZABLE)
- Omówić dodatkowy poziom izolacji wprowadzony przez MS SQL Server od wersji 2005
    - Migawka (SNAPSHOT)
- Różne poziomy izolacji wpływają na pojawienie się lub nie różnych anomalii.
- Stworzyć macierz możliwości występowania anomalii w zależności od zastosowanego poziomu izolacji bazy danych.
- Aby zapewnić odpowiedni poziom izolacji silniki baz danych stosują blokady różnego poziomu, proszę omówić typowe blokady zakładane przez DBMS, wskazując główne różnicę oraz skutki ich używania w praktyce.
- Omówić zjawisko zakleszczenia (deadlocks), co to jest, w jakich sytuacjach występuje i czy istnieją metody pozwalające programiście na zminimalizowanie ryzyka ich wystąpienia, jeżeli tak to jakie.
- Omówić typ danych timestamp ewentualnie rowversion – UWAGA omówienie należy zrobić dla bazy danych MS SQL. Wskazać czym się różni typ timestamp w MS SQL od timestamp w MySQL.

- Rozwinąć skrót ORM oraz postarać się opisać idee stosowania rozwiązań ORM.
- Szerzej opisać zasadnicze różnice pomiędzy dotychczas omawianym dostępem z wykorzystaniem ADO.NET względem innych rozwiązań ORM (w uogólnieniu bez omawiania szczegółowych różnic implementacyjnych pomiędzy różnymi Frameworkami).
- Dokonać porównania pomiędzy różnymi rozwiązaniami ORM (wskazać cechy wspólne oraz różnice), w szczeglności proszę się skupić na:
    - Entity Framework
    - LINQ to SQL
    - NHibernate
    - Dapper
- Na podstawie pierwszej analizy porównawczej wskazać który z powyższych ORM, w Państwa ocenie na „pierwszy rzut oka" stwarza wrażenie najodpowiedniejszego do dalszej pracy, odpowiedź postarać się uzasadnić.
- Omówić poszczególne podejścia oraz różnice pomiędzy nimi:
    - Database-First
    - Code-First
    - Model-First
- Na podstawie omówienia różnić pomiędzy podejściami proszę również spróbować wskazać które z powyższych, w Państwa ocenie na „pierwszy rzut oka" stwarza wrażenie najodpowiedniejszego do dalszej pracy, odpowiedź postarać się uzasadnić.
- Czym są Migracje, gdzie są wykorzystywane oraz jaki jest cel ich stosowania.
- Opisać oraz dokonać porównania pomiędzy Lazy loading i Eager loading, w przypadku EF które z powyższych jest niejako naturalne (domyślne), co trzeba zrobić by jednak „zmusić" EF do wykorzystania "drugiej opcji"?
- Omówić czym w ORM jest Raw SQL Query, w jakich przypadkach użycie RAW Query może być uzasadnione?

6. Przebieg ćwiczenia praktycznego

Wykorzystując bazę danych (tabelę) z danymi, utworzoną w trakcie realizacji Laboratorium 1 - „Kody_Pocztowe" zawierającą następujące kolumny: **Kod_Pocztow**y, **Adres**, **Miejscowosc**, **Wojewodztwo**, **Powiat**. Utworzyć prosty formularz zawierający

prezentowane dane (na kontrolce DataGridView) oraz opisane pola wyszukiwania (texBox) w których można, ale nie trzeba wprowadzić warunki wyszukiwania po kryterium „zawiera", oraz przycisk, po przyciśnięciu którego zaprezentowane zostaną wszystkie informację spełniające zadane kryterium/kryteria wyszukiwania. Przy czym kryteria te mogą dotyczyć jednej/kilku lub wszystkich kolumn. Można wykorzystać, zapytanie linku w klauzuli „where().toList()", rawSql, lub w ostateczności dokonać filtrowania i wyszukania na podstawie listy zawierającej wszystkie dane. Oczywiście możliwym jest zaproponowanie wszystkich trzech rozwiązań w ramach jednego projektu: wyszukiwanie po trzech metodach wyzwalane z oddzielnych przycisków, w tym przypadku można też dokonać porównania czasów realizacji zapytań w zależności od wybranej metody.

Jakie będą zasadnicze różnice pomiędzy „wybraniem" wyników z bazy danych na podstawie wykorzystania klauzuli „where(...).toList()" lub rawSql względem zapytania zwracającego wszystkie dane i filtrujące je w pamięci operacyjnej komputera w trakcie działania aplikacji ?

7. Opracowanie sprawozdania

Z racji założonych wytycznych laboratoryjnych opracowanie sprawozdania zostało podzielone na dwie części, teoretyczną i praktyczną.

Teoretyczna część zawiera rozwinięcia pojęć, a także wyjaśnienie ich na własne rozumienie,

Praktyczna część zawiera realizację programu, w którym zostały pokazane metody wyszukiwania i filtrowania za pomocą ADO.NET, Entity Framework i Dapper.

7.1 Opracowanie teoretyczne

Rozwinąć anagram ACID, wypisać oraz opisać znaczenie słów składowych anagramu w kontekście przetwarzania transakcyjnego

ACID w kontekście przetwarzania transakcyjnego oznacza:

Atomicity (Atomowość): Transakcja jest wykonywana w całości lub wcale.

Consistency (Spójność): Transakcja musi przestrzegać reguł integralności danych.

Isolation (Izolacja): Transakcje są niezależne od siebie i nie zakłócają się nawzajem.

Durability (Trwałość): Zmiany wprowadzone przez transakcję są trwałe, nawet w przypadku awarii systemu.

Te zasady zapewniają spójność, niezależność i trwałość operacji transakcyjnych w bazach danych

Omówić typowe anomalie transakcji jakie mogą wystąpić: Brudny odczyt (Dirty read), Utracona modyfikacja (Lost update), Niepowtarzalny odczyt (Non-repeatable read), Fantomy (Phantoms)

Anomalia brudnego odczytu występuje, gdy jedna transakcja odczytuje dane, które zostały tymczasowo zmienione przez inną transakcję, ale nie zostały jeszcze zatwierdzone (czyli nie są trwałe). Jeśli transakcja, która wprowadziła te zmiany, zostanie wycofana (rollback), to inne transakcje odczytują nieprawdziwe dane.

Anomalia utraconej modyfikacji występuje, gdy dwie lub więcej transakcje równocześnie modyfikują te same dane, a jedna z nich nadpisuje zmiany wprowadzone przez inną transakcję. W rezultacie, jedna z aktualizacji jest tracona, a zmiany nie są zachowane w sposób spójny.

Anomalia niepowtarzalnego odczytu występuje, gdy ta sama transakcja odczytuje te same dane więcej niż raz, ale otrzymuje różne wartości podczas każdego odczytu. To zdarza się, gdy inna transakcja wprowadza zmiany w odczytywanych danych między dwoma odczytami, co prowadzi do niespójności odczytu.

Anomalia fantomowego odczytu występuje, gdy transakcja wykonuje zapytanie o zestaw danych, a następnie wykonuje to samo zapytanie ponownie, ale otrzymuje różny zestaw danych, ponieważ inne transakcje wprowadziły nowe wiersze, które spełniają warunki zapytania. Jest to podobne do niespójnego odczytu, ale w przypadku fantomowego odczytu zmieniają się nie tylko wartości kolumn, ale także liczba wierszy.

Omówić 4 poziomy izolacji według standardu SQL-92: Niezatwierdzony Odczyt (READ UNCOMMITTED), Odczyt Zatwierdzonych Danych (READ COMMITTED), Powtarzalny Odczyt (REPEATABLE READ), Uszeregowany (SERIALIZABLE)

W „Niezatwierdzony Odczyt (READ UNCOMMITTED)” poziomie izolacji transakcji, znanej również jako "brudne odczyty" (dirty read), transakcje mogą odczytywać niezatwierdzone zmiany wprowadzone przez inne transakcje. Oznacza to, że transakcja może odczytać

tymczasowe wartości, które mogą zostać cofnięte przez inną transakcję. Ten poziom izolacji nie gwarantuje spójności ani poprawności danych.

W „Odczyt Zatwierdzonych Danych (READ COMMITTED):" poziomie izolacji, transakcje odczytują tylko zatwierdzone zmiany wprowadzone przez inne transakcje. Oznacza to, że dane widoczne dla danej transakcji są spójne i zatwierdzone przez inne transakcje w momencie odczytu. Jednak między dwoma odczytami tej samej transakcji mogą wystąpić niespójne dane, jeśli inne transakcje dokonują zmian.

W „Powtarzalny Odczyt (REPEATABLE READ)" poziomie izolacji, transakcje odczytują tylko dane, które były obecne w momencie rozpoczęcia transakcji. Oznacza to, że wszystkie odczyty dokonane przez tę samą transakcję zwrócą takie same wyniki, nawet jeśli inne transakcje wprowadzą zmiany w tych danych. Zapobiega to anomalii niespójnego odczytu.

W „Uszeregowany (SERIALIZABLE)" poziomie izolacji, transakcje są izolowane w pełni, a żadne anomalia transakcji nie występują. Oznacza to, że żadna inna transakcja nie może modyfikować danych, które są odczytywane przez bieżącą transakcję, a także żadna inna transakcja nie może odczytywać danych, które są zmieniane przez bieżącą transakcję. Ten poziom izolacji gwarantuje najwyższą spójność danych, ale może prowadzić do blokowania i konfliktów między transakcjami.

Omówić dodatkowy poziom izolacji wprowadzony przez MS SQL Server od wersji 2005: Migawka (SNAPSHOT)

W poziomie izolacji Migawki, każda transakcja odczytuje dane tak, jakby były zarejestrowane w momencie rozpoczęcia transakcji. Oznacza to, że transakcje nie będą blokować się nawzajem podczas odczytu i zapisu danych. Jest to możliwe dzięki mechanizmowi tworzenia i przechowywania migawek danych.

Różne poziomy izolacji wpływają na pojawienie się lub nie różnych anomalii? Stworzyć macierz możliwości występowania anomalii w zależności od zastosowanego poziomu izolacji bazy danych.

Tak, różne poziomy izolacji mają wpływ na pojawienie się lub nie różnych anomalii transakcji. Wyższe poziomy izolacji zazwyczaj zapewniają większą spójność danych kosztem wydajności i blokowania się transakcji. Niższe poziomy izolacji mogą zwiększać wydajność, ale mogą prowadzić do nieprawidłowych wyników lub niespójności danych.

- Anomalia utraconej aktualizacji:
  - READ UNCOMMITTED: Może występować.
  - READ COMMITTED: Może występować.
  - REPEATABLE READ: Nie występuje.
  - SERIALIZABLE: Nie występuje.
- Anomalia brudnego odczytu:
  - READ UNCOMMITTED: Występuje.
  - READ COMMITTED: Nie występuje.
  - REPEATABLE READ: Nie występuje.
  - SERIALIZABLE: Nie występuje.
- Anomalia niespójnego odczytu:
  - READ UNCOMMITTED: Występuje.
  - READ COMMITTED: Występuje.
  - REPEATABLE READ: Nie występuje.
  - SERIALIZABLE: Nie występuje.
- Anomalia fantomowego odczytu:
  - READ UNCOMMITTED: Występuje.
  - READ COMMITTED: Występuje.
  - REPEATABLE READ: Występuje.
  - SERIALIZABLE: Nie występuje.

Aby zapewnić odpowiedni poziom izolacji silniki baz danych stosują blokady różnego poziomu, proszę omówić typowe blokady zakładane przez DBMS, wskazując główne różnicę oraz skutki ich używania w praktyce.

1. Blokady wiersza (Row Locks):
   a. **Różnica**: Blokady wiersza zakładane są na poziomie indywidualnych wierszy w tabeli.
   b. **Skutki**: Zapewniają wyższy poziom równoległości, ponieważ tylko konkretne wiersze są blokowane, umożliwiając równoczesne modyfikacje innych wierszy w tabeli. Jednak może prowadzić do większej liczby konfliktów i blokowań, szczególnie w przypadku dużych operacji aktualizacji danych.
2. Blokady strony (Page Locks):

   a. **Różnica**: Blokady strony zakładane są na poziomie grupy wierszy w tabeli, zwykle na poziomie strony danych (np. 4 KB).
   b. **Skutki**: Może zredukować liczbę konfliktów i blokowań w porównaniu do blokad wiersza, ponieważ większa liczba wierszy jest blokowana jednocześnie. Jednak może również wprowadzać wyższy narzut zarządzania blokadami i ograniczać równoległość, gdy inne transakcje chcą modyfikować różne wiersze w tej samej stronie danych.
3. Blokady tabeli (Table Locks):
   a. **Różnica**: Blokady tabeli zakładane są na poziomie całej tabeli, blokując dostęp do wszystkich wierszy w tabeli.
   b. **Skutki**: Jest to najbardziej restrykcyjny rodzaj blokady, ponieważ uniemożliwia równoczesny dostęp do jakichkolwiek danych w tabeli przez inne transakcje. Może prowadzić do blokowania się i spowolnienia przetwarzania w przypadku dużej liczby transakcji jednocześnie próbujących uzyskać dostęp do tabeli.

Omówić zjawisko zakleszczenia (deadlocks), co to jest, w jakich sytuacjach występuje i czy istnieją metody pozwalające programiście na zminimalizowanie ryzyka ich wystąpienia, jeżeli tak to jakie.

Zakleszczenie (deadlock) jest sytuacją, w której dwa lub więcej wątków lub procesów wzajemnie blokują się nawzajem, oczekując na zasoby, które są już zajęte przez inne wątki lub procesy. W wyniku zakleszczenia żadne z blokujących się zasobów nie zostaje zwolnione, co prowadzi do zatrzymania postępu w systemie i niewykonania dalszych operacji.

Zakleszczenie może wystąpić w takich sytuacjach, gdy zachodzą cztery warunki naraz:

- Wzajemne wykluczanie (Mutual Exclusion): Co najmniej jeden zasób jest zarezerwowany wyłącznie dla jednego wątku lub procesu i nie może być jednocześnie używany przez inne.
- Posiadanie i oczekiwanie (Hold and Wait): Wątek lub proces trzyma przynajmniej jeden zasób i oczekuje na inne zasoby, które są zajęte przez inne wątki lub procesy.

- Brak wywłaszczania (No Preemption): Zasoby nie mogą być wywłaszczane siłą od wątków lub procesów, które je trzymają. Zasoby mogą być zwalniane tylko dobrowolnie.
- Oczekiwanie na cykl (Circular Wait): Istnieje cykliczna zależność między dwoma lub więcej wątkami lub procesami, gdzie każdy z nich oczekuje na zasób, który jest trzymany przez inny wątek lub proces.

Sytuacje, w których często występuje zakleszczenie, to:

- Konkurencyjny dostęp do wspólnych zasobów, takich jak bazy danych, pliki, pamięć współdzielona itp.
- Wykorzystywanie blokad przez wątki lub procesy bez odpowiedniej koordynacji.
- Niespójne lub niewłaściwe zarządzanie kolejnością blokad.

Istnieją różne metody pozwalające na minimalizowanie ryzyka wystąpienia zakleszczenia:

- Hierarchiczne przydzielanie zasobów: Upewnienie się, że zasoby są przydzielane w hierarchiczny sposób, zapobiegając cyklicznym zależnościom.
- Unikanie blokowania z wykorzystaniem algorytmów bezblokowych: Zastosowanie technik bezblokowych, które eliminują potrzebę blokowania.
- Unikanie czekania na zasoby: Jeśli wątek nie może uzyskać dostępu do zasobu, może go zwolnić i spróbować ponownie później, zamiast czekać w nieskończoność.
- Ustalanie porządku blokowania: Zastosowanie ustalonego porządku blokowania dla zasobów.

Omówić typ danych timestamp ewentualnie rowversion – UWAGA omówienie należy zrobić dla bazy danych MS SQL. Wskazać czym się różni typ timestamp w MS SQL od timestamp w MySQL.

Typ timestamp w MS SQL Server jest używany jako mechanizm kontroli wersji do wykrywania zmian w wierszach, podczas gdy timestamp w MySQL służy do przechowywania wartości daty i godziny. Różnią się zarówno reprezentacją danych, jak i przeznaczeniem.

Rozwinąć skrót ORM oraz postarać się opisać idee stosowania rozwiązań ORM.

ORM (Object-Relational Mapping), czyli mapowanie obiektowo-relacyjne. Jest to technika programowania, która umożliwia programistom pracę z danymi w relacyjnej bazie danych za pomocą obiektów w językach programowania obiektowego

Idea stosowania rozwiązań ORM wynika z próby ułatwienia i usprawnienia procesu interakcji między relacyjnymi bazami danych a kodem programu. Tradycyjne metody komunikacji z bazą danych, takie jak pisanie ręcznych zapytań SQL i mapowanie wyników na obiekty, mogą być skomplikowane, czasochłonne i podatne na błędy. ORM ma na celu uproszczenie tego procesu poprzez automatyczne mapowanie obiektów na tabele w bazie danych i umożliwienie programistom manipulacji danymi za pomocą obiektów, a nie bezpośrednio przez zapytania SQL.

Szerzej opisać zasadnicze różnice pomiędzy dotychczas omawianym dostępem z wykorzystaniem ADO.NET względem innych rozwiązań ORM (w uogólnieniu bez omawiania szczegółowych różnic implementacyjnych pomiędzy różnymi Frameworkami).

ADO.NET jest nisko-poziomowym interfejsem dostępu do danych, który skupia się na bezpośredniej obsłudze połączenia z bazą danych, tworzeniu zapytań SQL i zarządzaniu danymi w sposób bardziej bezpośredni. W przeciwieństwie do tego, rozwiązania ORM dostarczają wyższy poziom abstrakcji, który umożliwia pracę z danymi na poziomie obiektów, eliminując konieczność pisania ręcznych zapytań SQL.

ADO.NET jest bardziej skomplikowane w użyciu niż rozwiązania ORM. Wymaga większej ilości kodu i większego wysiłku programisty w celu wykonania prostych operacji, takich jak pobieranie danych, aktualizowanie rekordów, zarządzanie relacjami itp. Rozwiązania ORM dostarczają bardziej intuicyjnego API i skraca czas oraz wysiłek potrzebny do pisania kodu dostępu do danych.

ADO.NET wymaga ręcznego zarządzania połączeniem z bazą danych oraz transakcjami. Programista musi otwierać i zamykać połączenia, rozpoczynać i zatwierdzać transakcje. Natomiast rozwiązania ORM automatycznie zarządzają tymi aspektami, zapewniając wygodne i bezpieczne zarządzanie połączeniem i transakcjami.

ADO.NET jest specyficzne dla platformy .NET i zazwyczaj jest używane tylko w środowiskach opartych na tej platformie. Z kolei rozwiązania ORM są często dostępne dla różnych języków programowania i platform, co umożliwia przenośność kodu między różnymi systemami.

Dokonać porównania pomiędzy różnymi rozwiązaniami ORM (wskazać cechy wspólne oraz różnice), w szczególności proszę się skupić na Entity Framework, LINQ to SQL, NHibernate, Dapper. Na podstawie pierwszej analizy porównawczej wskazać który z powyższych ORM, w Państwa ocenie na „pierwszy rzut oka" stwarza wrażenie najodpowiedniejszego do dalszej pracy, odpowiedź postarać się uzasadnić.

1. Entity Framework (EF):

- Cechy wspólne:
  - W pełni obsługuje mapowanie obiektowo-relacyjne (ORM).
  - Dostarcza bogate API do manipulowania danymi, tworzenia zapytań i zarządzania relacjami.
  - Wsparcie dla różnych dostawców baz danych.
  - Obsługa transakcji, zarządzania połączeniami i śledzenia zmian w obiektach.
- Różnice:
  - Wyższy poziom abstrakcji niż LINQ to SQL i Dapper.
  - W pełni zintegrowany z innymi funkcjami platformy .NET.
  - Obsługa zaawansowanych funkcji, takich jak dziedziczenie, asocjacje, dziedziczenie tabel itp.

2. LINQ to SQL:

- Cechy wspólne:
  - Mapowanie obiektowo-relacyjne (ORM) z wykorzystaniem języka zapytań LINQ.
  - Prosta konfiguracja i używanie.
  - Wsparcie dla większości dostawców baz danych.
- Różnice:
  - Mniejsza funkcjonalność w porównaniu z Entity Framework.
  - Bardziej skoncentrowane na prostych scenariuszach dostępu do danych.

3. NHibernate:

- Cechy wspólne:
  - Pełne mapowanie obiektowo-relacyjne (ORM) i obsługa języka zapytań HQL.
  - Znaczna elastyczność i konfigurowalność.
  - Obsługa różnych baz danych.

- Różnice:
    - Większa skomplikowana konfiguracja niż Entity Framework i LINQ to SQL.
    - Wymaga więcej kodu w porównaniu z Entity Framework.

4. Dapper:

- Cechy wspólne:
    - Proste i lekkie rozwiązanie ORM.
    - Wydajne mapowanie danych do obiektów.
    - Minimalny narzut na wydajność.
- Różnice:
    - Mniejsza funkcjonalność w porównaniu z Entity Framework, LINQ to SQL i NHibernate.
    - Bardziej skupione na bezpośrednim mapowaniu danych, bez zaawansowanych funkcji ORM.

Wybór najlepszego ORM zależy od specyficznych potrzeb i kontekstu projektu. Jednak na podstawie analizy porównawczej, Entity Framework wydaje się być najodpowiedniejszym rozwiązaniem do dalszej pracy. Posiada szerokie wsparcie ze strony Microsoft i jest pełnowartościowym ORM z zaawansowanymi funkcjami.

Omówić poszczególne podejścia oraz różnice pomiędzy nimi: Database-First, Code-First, Model-First. Na podstawie omówienia różnić pomiędzy podejściami proszę również spróbować wskazać które z powyższych, w Państwa ocenie na „pierwszy rzut oka" stwarza wrażenie najodpowiedniejszego do dalszej pracy, odpowiedź postarać się uzasadnić.

W podejściu Database-First najpierw istnieje istniejąca baza danych, która jest centralnym punktem projektu. Na podstawie istniejącej bazy danych, ORM generuje automatycznie modele obiektowe (klasy) na podstawie struktury i schematu bazy danych. Programista następnie rozszerza modele obiektowe o dodatkową logikę biznesową, jeśli jest to wymagane. To podejście jest przydatne, gdy baza danych już istnieje i chcemy skorzystać z istniejącego schematu.

W podejściu Code-First programista definiuje modele obiektowe (klasy) w języku programowania, bez konieczności istnienia bazy danych. Na podstawie tych modeli, ORM generuje schemat bazy danych, tworząc tabele, relacje itp. Programista ma pełną kontrolę nad strukturą bazy danych poprzez definiowanie modeli obiektowych i konfigurację, to podejście jest

użyteczne, gdy chcemy zacząć od modelu obiektowego i automatycznie wygenerować schemat bazy danych.

W podejściu Model-First projekt rozpoczyna się od tworzenia diagramu lub modelu graficznego, który reprezentuje strukturę danych i relacje. Na podstawie tego modelu, ORM generuje zarówno modele obiektowe, jak i schemat bazy danych. Dostosowany może być generowany kod lub struktura bazy danych na podstawie potrzeb. To podejście jest użyteczne, gdy preferowane jest wizualne projektowanie modelu danych przed implementacją.

Przeze mnie wykorzystywana w równej skali jak metoda Code-First tak i Database-First i najbardziej to zależy od tego z czym jest rozpoczynana praca, czy to projekt, który startuje od zera, czy projekt, który posiada już swoją bazę informacyjną.

Czym są Migracje, gdzie są wykorzystywane oraz jaki jest cel ich stosowania.

Migracje w kontekście ORM to mechanizm, który umożliwia automatyczne zarządzanie ewolucją schematu bazy danych wraz z postępem projektu. Są one wykorzystywane głównie w podejściu Code-First, choć niektóre narzędzia ORM oferują również migracje w innych podejściach. Głównym celem migracji jest utrzymanie spójności pomiędzy modelem obiektowym a strukturą bazy danych, zapewniając elastyczność w rozwoju aplikacji. W miarę ewoluowania projektu, mogą zachodzić zmiany w modelu danych, takie jak dodawanie nowych klas, zmiana nazw, dodawanie nowych pól lub relacji. Migracje pozwalają na automatyczne aktualizowanie schematu bazy danych w sposób kontrolowany i bez utraty danych.

Opisać oraz dokonać porównania pomiędzy Lazy loading i Eager loading, w przypadku EF które z powyższych jest niejako naturalne (domyślne), co trzeba zrobić by jednak „zmusić" EF do wykorzystania "drugiej opcji"?

Lazy loading - powiązane dane są wczytywane dopiero w momencie, gdy są potrzebne. Domyślne w Entity Framework. Eager loading - powiązane dane są wczytywane razem z głównym obiektem, gdy jest on wczytywany. Wymaga jawnego określenia, które dane mają być wczytane.

Lazy loading jest wygodne, ale może prowadzić do wielu zapytań do bazy danych. Eager loading jest bardziej kontrolowane i efektywne, ale wymaga określenia wczytywanych danych. Aby użyć eager loadingu w Entity Framework, należy jawnie określić powiązane obiekty, które mają być wczytane, za pomocą metody Include() w zapytaniu LINQ lub konfiguracji relacji.

Omówić czym w ORM jest Raw SQL Query, w jakich przypadkach użycie RAW Query może być uzasadnione?

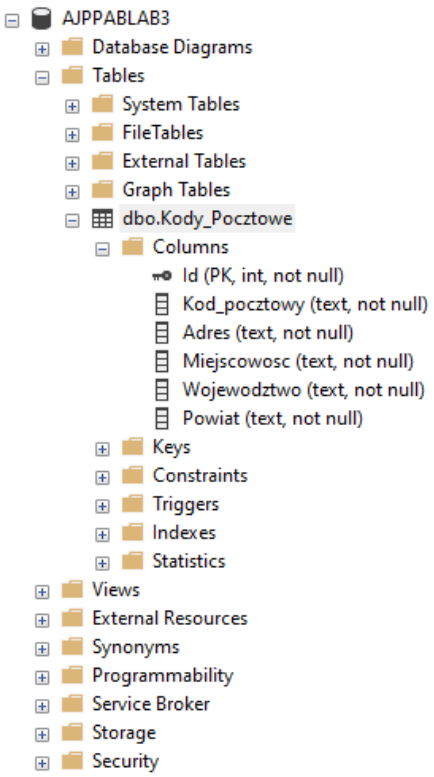
Raw SQL Query w ORM to mechanizm, który umożliwia wykonywanie surowych zapytań SQL bezpośrednio wewnątrz kodu, zamiast korzystania z konstrukcji zapytań ORM.

Użycie Raw SQL Query może być uzasadnione w następujących przypadkach:

- Skomplikowane zapytania
- Optymalizacja wydajności
- Specyficzne funkcje bazodanowe
- Migracje i zarządzanie schematem

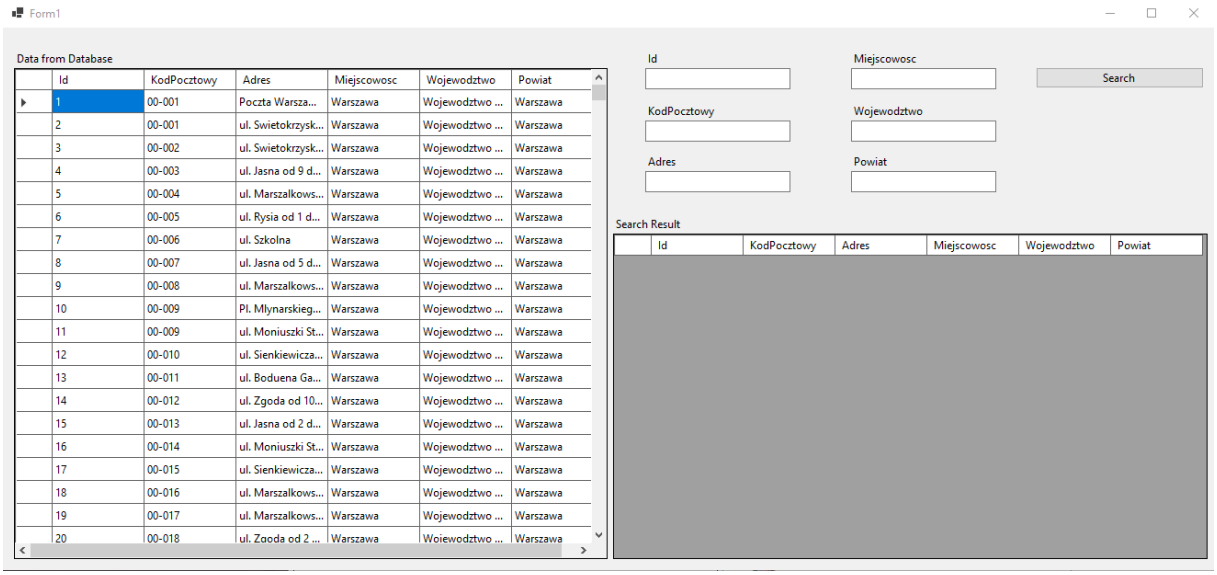
7.2 Opracowanie praktyczne

Wykorzystana została baza danych z kodami pocztowymi, która była utworzona w trakcie realizacji Laboratorium nr. 1, i która zawiera takie kolumny jak Kod_Pocztowy, Adres, Miejscowosc, Wojewodztwo, Powiat. Strukturę tej bazy danych można zobaczyć na *Rysunek 1*

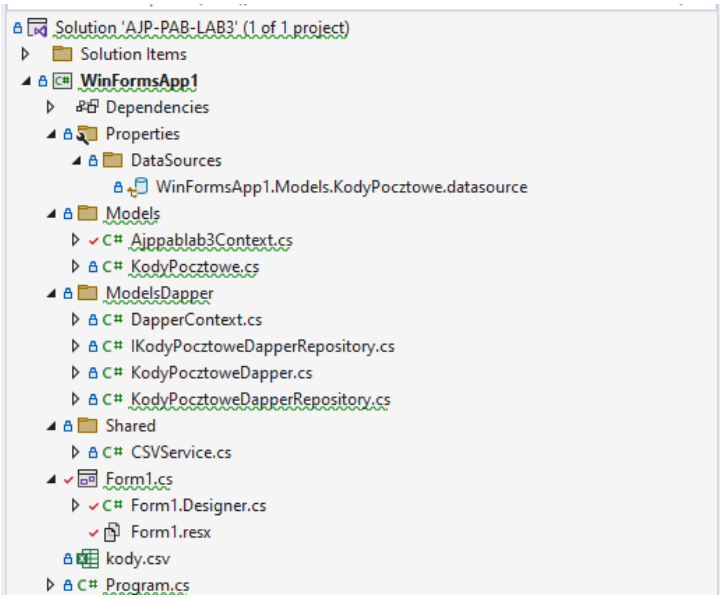


Rysunek 1. Struktura bazy danych z kodami pocztowymi

Do realizacji laboratorium został utworzony projekt Windows Forms .NET7 z formularzem, który jest pokazany na *Rysunek 2* i struktura samego projektu jest pokazana na *Rysunek* 3. Form1 składa się z takich kontrolek jak: DataGridView do wyświetlania całej tabeli, DataGridView do wyświetlania wyników wyszukiwania, TextBox'y do wprowadzania kryteriów wyszukiwania, Label'y do oznacznia TextBox'ó, Button'y do wybrania wyszukiwania odpowiednią metodą.



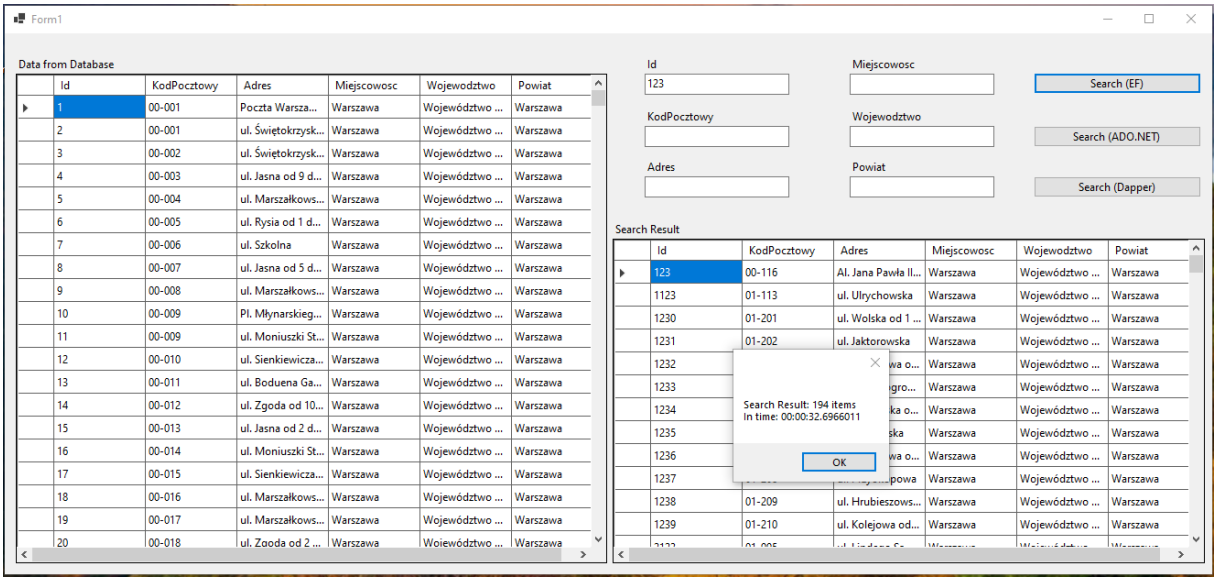
Rysunek 2. Wygląd programu



Rysunek 3. Struktura projektu

Następnie zostały utworzony trzy metody wyszukiwania dla trzech przycisków: Entity Framework, ADO.NET i Dapper.

Na *Rysunek 4* jest pokazany wynik działania metody korzystającej z Entity Framework, a na *Snippet 2* jest pokazany kod realizującą tą funkcjonalność. Po kluczu „123” zostały odnalezione 194 wiersze w 32 sekundy.



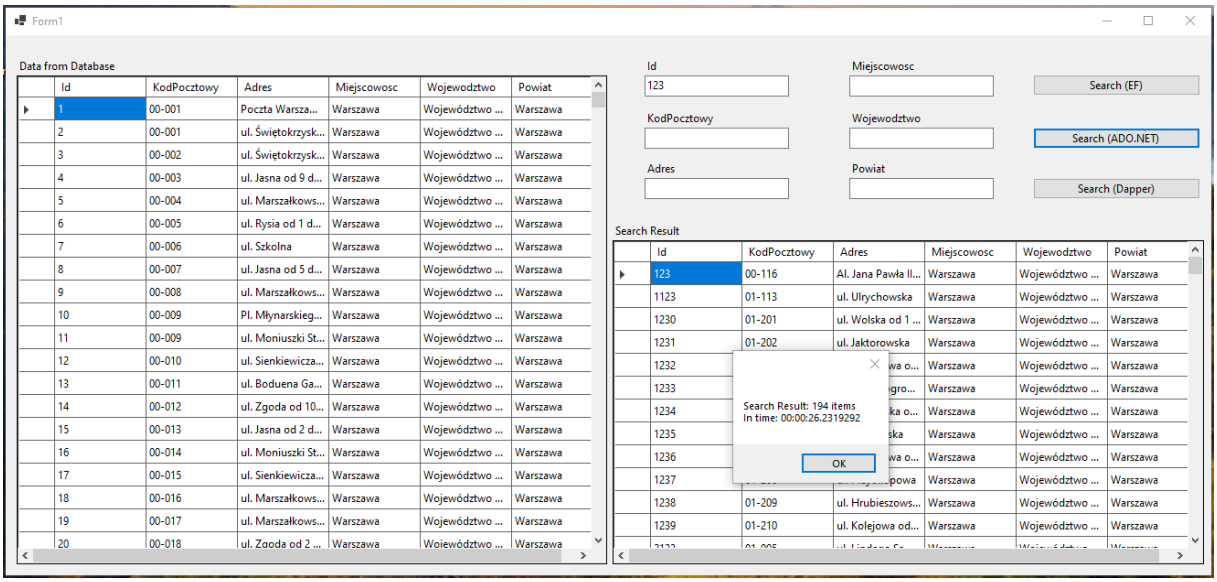
Rysunek 4. Wyszukanie za pomocą Entity Framework

Snippet 1. Metoda wyszukiwania za pomocą EF

```
private void efSearchButton_Click(object sender, EventArgs e)
{
    try
    {
        string? id = idTextBox.Text.Length > 0 ? idTextBox.Text : null;
        string? miejscowosc = miejscowoscTextBox.Text.Length > 0 ? miejscowoscTextBox.Text : null;
        string? powiat = powiatTextBox.Text.Length > 0 ? powiatTextBox.Text : null;
        string? adres = adresTextBox.Text.Length > 0 ? adresTextBox.Text : null;
        string? kodPocztowy = kodPocztowyTextBox.Text.Length > 0 ? kodPocztowyTextBox.Text : null;
        string? wojewodztwo = wojewodztwoTextBox.Text.Length > 0 ? wojewodztwoTextBox.Text : null;

        List<KodyPocztowe> result = new();
        Stopwatch stopwatch = new();
        stopwatch.Start();
        using (AjppablabContext db = new())
        {
            result = db.KodyPocztowes.ToList();
            if (id != null)
            {
                result = result.Where(kod => kod.Id.ToString().ToLower().Contains(id.ToLower())).ToList();
            }
            if (miejscowosc != null)
            {
                result = result.Where(kod => kod.Miejscowosc.ToLower().Contains(miejscowosc.ToLower())).To-
List();
            }
            if (powiat != null)
            {
                result = result.Where(kod => kod.Powiat.ToLower().Contains(powiat.ToLower())).ToList();
            }
            if (adres != null)
            {
                result = result.Where(kod => kod.Adres.ToLower().Contains(adres.ToLower())).ToList();
            }
            if (kodPocztowy != null)
            {
                result = result.Where(kod => kod.KodPocztowy.ToLower().Contains(kodPocztowy.ToLower())).To-
List();
            }
            if (wojewodztwo != null)
            {
                result = result.Where(kod => kod.Wojewodztwo.ToLower().Contains(wojewodztwo.ToLower())).To-
List();
```

```
            }
            if (result.Count < 0)
            {
                throw new Exception("Nothing Found");
            }
        }
        stopwatch.Stop();

        resultDataGridView.DataSource = result;
        MessageBox.Show($"Search Result: {result.Count} items \nIn time: {stopwatch.Elapsed}");
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Occured error {ex.Message}");
    }
}
```

Na *Rysunek 5* jest pokazany wynik działania metody korzystającej z ADO.NET, a *Snippet* 2 jest pokazany kod realizującą tą funkcjonalność. Po kluczu „123” zostały odnalezione 194 wiersze w 26 sekund.



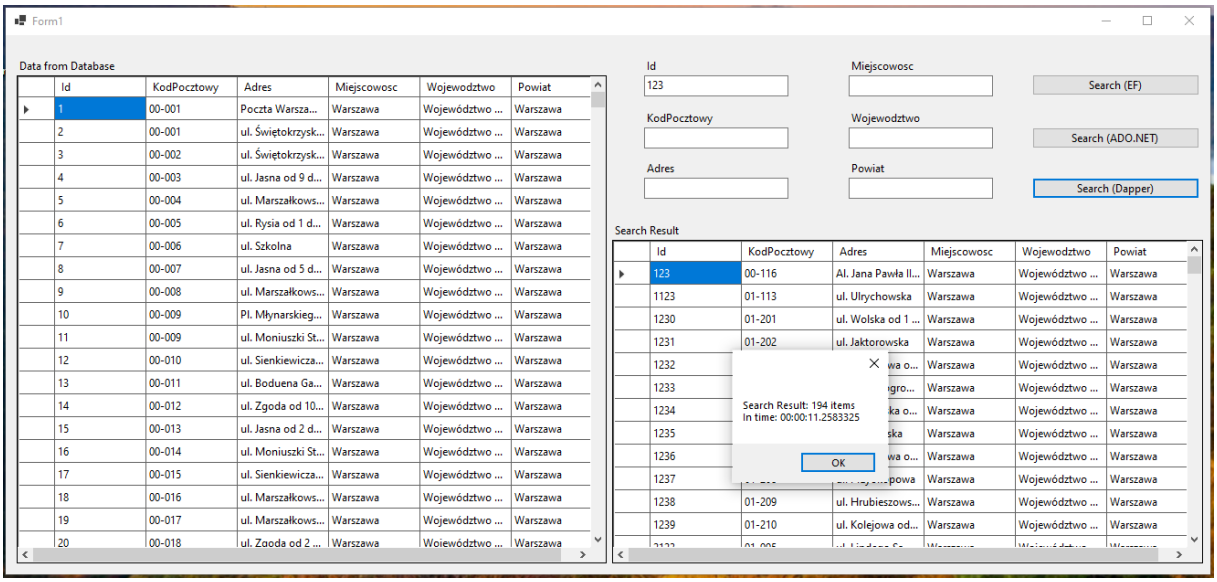
Rysunek 5. Wyszukanie za pomocą ADO.NET

Snippet 2. Metoda wyszukiwania za pomocą ADO.NET

```
private void adonetSearchButton_Click(object sender, EventArgs e)
{
    try
    {
        int? id = null;
        if (Int32.TryParse(idTextBox.Text, out _)) id = Int32.Parse(idTextBox.Text);

        string? miejscowosc = miejscowoscTextBox.Text.Length > 0 ? miejscowoscTextBox.Text : null;
        string? powiat = powiatTextBox.Text.Length > 0 ? powiatTextBox.Text : null;
        string? adres = adresTextBox.Text.Length > 0 ? adresTextBox.Text : null;
        string? kodPocztowy = kodPocztowyTextBox.Text.Length > 0 ? kodPocztowyTextBox.Text : null;
        string? wojewodztwo = wojewodztwoTextBox.Text.Length > 0 ? wojewodztwoTextBox.Text : null;

        DataView dataView = new();
        Stopwatch stopwatch = new();
        stopwatch.Start();
        using (SqlConnection connection = new(connectionString))
        {
            connection.Open();
            SqlCommand command = new();
            command.Connection = connection;
            command.CommandText = "SELECT Id, Kod_pocztowy AS KodPocztowy, Adres, Miejscowosc, Wojewodztwo,
Powiat FROM Kody_Pocztowe";
```

```
            SqlDataReader reader = command.ExecuteReader();

            DataTable dataTable = new DataTable();
            dataTable.Load(reader);
            dataTable.CaseSensitive = false;
            dataView = dataTable.DefaultView;
            string filtr = "";

            if (id != null)
            {
                filtr += $"(CONVERT(Id, System.String) LIKE '%{id}%')";
            }
            if (miejscowosc != null)
            {
                filtr += (filtr.Length > 0 ? $" AND (Miejscowosc LIKE '%{miejscowosc}%')" : $"(Miejscowosc
LIKE '%{miejscowosc}%')");
            }
            if (powiat != null)
            {
                filtr += (filtr.Length > 0 ? $" AND (Powiat LIKE '%{powiat}%')" : $"(Powiat LIKE '%{po-
wiat}%')");
            }
            if (adres != null)
            {
                filtr += (filtr.Length > 0 ? $" AND (Adres LIKE '%{adres}%')" : $"(Adres LIKE '%{ad-
res}%')");
            }
            if (kodPocztowy != null)
            {
                filtr += (filtr.Length > 0 ? $" AND (KodPocztowy LIKE '%{kodPocztowy}%')" : $"(KodPocztowy
LIKE '%{kodPocztowy}%')");
            }
            if (wojewodztwo != null)
            {
                filtr += (filtr.Length > 0 ? $" AND (Wojewodztwo LIKE '%{wojewodztwo}%')" : $"(Wojewodztwo
LIKE '%{wojewodztwo}%')");
            }

            dataView.RowFilter = filtr;

            if (dataView.Count < 0)
            {
                throw new Exception("Nothing Found");
            }
        }
        stopwatch.Stop();

        resultDataGridView.DataSource = dataView;
        MessageBox.Show($"Search Result: {dataView.Count} items \nIn time: {stopwatch.Elapsed}");
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Occured error {ex.Message}");
    }
}
```

Na *Rysunek 6* jest pokazany wynik działania metody korzystającej z Dapper'a, a na *Snippet 3 Snippet 4* jest pokazany kod realizującą tą funkcjonalność. Po kluczu „123" zostały odnalezione 194 wiersze w 11 sekund, co jest najlepszym wynikiem z tych trzech metod.



Rysunek 6. Wyszukanie za pomocą Dapper

Snippet 3. Metoda wyszukiwania za pomocą Dapper (1)

```
private void dapperSearchButton_Click(object sender, EventArgs e)
{
    try
    {
        int? id = null;
        if (Int32.TryParse(idTextBox.Text, out _)) id = Int32.Parse(idTextBox.Text);

        string? miejscowosc = miejscowoscTextBox.Text.Length > 0 ? miejscowoscTextBox.Text : null;
        string? powiat = powiatTextBox.Text.Length > 0 ? powiatTextBox.Text : null;
        string? adres = adresTextBox.Text.Length > 0 ? adresTextBox.Text : null;
        string? kodPocztowy = kodPocztowyTextBox.Text.Length > 0 ? kodPocztowyTextBox.Text : null;
        string? wojewodztwo = wojewodztwoTextBox.Text.Length > 0 ? wojewodztwoTextBox.Text : null;

        Stopwatch stopwatch = new();
        var result = new List<KodyPocztoweDapper>();
        stopwatch.Start();
        using (DapperContext dapperContext = new(connectionString))
        {
            KodyPocztoweDapperRepository kodyPocztoweDapperRepository = new(dapperContext);
            result = kodyPocztoweDapperRepository.GetKodyPocztowe(id, kodPocztowy, adres, miejscowosc, wo-
jewodztwo, powiat);
        }
        stopwatch.Stop();

        resultDataGridView.DataSource = result;
        MessageBox.Show($"Search Result: {result.Count()} items \nIn time: {stopwatch.Elapsed}");
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Occured error: {ex.Message}");
    }
}
```

Snippet 4. Metoda wyszukiwania za pomocą Dapper (2)

```
public class KodyPocztoweDapperRepository : IKodyPocztoweDapperRepository
{
    private readonly DapperContext _context;

    public KodyPocztoweDapperRepository(DapperContext context)
    {
        _context = context;
    }

    public List<KodyPocztoweDapper> GetKodyPocztowe(int? id, string? kodPocztowy, string? adres, string?
miejscowosc, string? wojewodztwo, string? powiat)
    {
        var filtr = "SELECT Id, Kod_pocztowy AS KodPocztowy, Adres, Miejscowosc, Wojewodztwo, Powiat FROM
Kody_Pocztowe";
```

```
        int counter = 0;

        if (id != null)
        {
            filtr += $" WHERE (CAST(CONVERT(VARCHAR(MAX), Id) AS INT) LIKE '%{id}%')";
            counter++;
        }
        if (miejscowosc != null)
        {
            filtr += (counter > 0 ? $" AND (Miejscowosc LIKE '%{miejscowosc}%')" : $" WHERE (Miejscowosc
LIKE '%{miejscowosc}%')");
            counter++;
        }
        if (powiat != null)
        {
            filtr += (counter > 0 ? $" AND (Powiat LIKE '%{powiat}%')" : $" WHERE (Powiat LIKE '%{po-
wiat}%')");
            counter++;
        }
        if (adres != null)
        {
            filtr += (counter > 0 ? $" AND (Adres LIKE '%{adres}%')" : $" WHERE (Adres LIKE '%{adres}%')");
            counter++;
        }
        if (kodPocztowy != null)
        {
            filtr += (counter > 0 ? $" AND (KodPocztowy LIKE '%{kodPocztowy}%')" : $" WHERE (KodPocztowy
LIKE '%{kodPocztowy}%')");
            counter++;
        }
        if (wojewodztwo != null)
        {
            filtr += (counter > 0 ? $" AND (Wojewodztwo LIKE '%{wojewodztwo}%')" : $" WHERE (Wojewodztwo
LIKE '%{wojewodztwo}%')");
            counter++;
        }

        using (var connection = _context.CreateConnection())
        {
            var kody_pocztowe = connection.Query<KodyPocztoweDapper>(filtr);
            return kody_pocztowe.ToList();
        }
    }
}
```

8. Wnioski

W trakcie laboratorium udało się poszerzyć wiedzę teoretyczną i praktyczną w zakresie wykorzystywania bibliotek ADO.NET i takich ORM jak Entity Framework i Dapper w tworzeniu aplikacji pracujących na ogromnej ilości danych.

Udało się przećwiczyć i zbadać różne sposoby wyszukiwania konkretnych informacji wśród dużych ilości danych za pomocą różnych technologii.

9. Bibliografia

1. Źródła pomocnicze
    a. Entity Framework Core Series [https://code-maze.com/entity-framework-core-series/], dostęp: 16.03.2023

b. Przewodnik po ADO.NET i pracy z bazami danych w .NET 6 (oryg. “Руководство по ADO.NET и работе с базами данных в .NET 6”) [https://metanit.com/sharp/adonetcore/], dostęp: 25.06.2023
c. Entity Framework Core 7 - Przewodnik (oryg. “Руководство по Entity Framework Core 7”) [https://metanit.com/sharp/efcore/], dostęp: 25.06.2023

2. Napotkane problem i ich rozwiązania
   a. Cannot perform 'Like' operation on System.Int32 and System.String. DataGridView search and filter [https://stackoverflow.com/questions/22328392/cannot-perform-like-operation-on-system-int32-and-system-string-datagridview], dostęp 16.03.2023

10. Spis ilustracji

11. Spis snippetów