Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження технологій доступу до реляційних баз даних
(тема)

Виконав:
Випускник 2 курсу, групи ІПЗм-21-1
Рамазанов Р.Ш.
(прізвище, ініціали)

Спеціальність 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми Освітньо-наукова
(код і повна назва спеціальності)

Керівник доц. Мазурова О.О.
(посада, прізвище)

Допускається до захисту
Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)
(повна назва)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітня-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програми Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРЖДУЮ:

Зав. кафедри _____
(підпис)

«___»_____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Рамазанову Расулу Шамільовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження технологій доступу до реляційних баз даних

Затверджена наказом університету 29.02.23 №302 ст

2. Термін подання студентом роботи до екзаменаційної комісії "_21_"_05__2023 р.

3. Вихідні данні до роботи *вимоги до розроблюваної програми, вимоги до архітектури системи, електронні ресурси за обраною темою, мови програмування C#, технології ASP.NET Core, СУБД MSSQL, середовища розробки Visual Studio, Microsoft SQL Management Studio.*

4. Перелік питань, що потрібно опрацювати в роботі *вступ, аналіз предметної області, формування вимог до програмного забезпечення, архітектура та проектування розробленого програмного забезпечення, опис прийнятих програмних рішень, тестування програмного забезпечення, висновки, додатки.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблемної області та постановка задачі	23.01 – 14.02.23	Виконано
2	Опис прийнятих проектних рішень	15.02 – 24.02.23	Виконано
3	Опис програмної реалізації	25.02 – 01.04.23	Виконано
4	Опис експериментальних досліджень	02.04 – 20.04.23	Виконано
5	Оформлення статті та тез доповіді	17.04 – 23.04.23	Виконано
6	Підготовка пояснювальної записки	01.04 – 26.04.23	Виконано
7	Підготовка презентації та доповіді	26.04 – 27.04.23	Виконано
8	Нормоконтроль	27.02 – 02.05.23	Виконано
9	Рецензування	03.05 – 06.05.23	Виконано
10	Занесення диплома в електронний архів	07.05.2023	Виконано
11	Попередній захист	07.05.2023	Виконано
12	Допуск до захисту у зав. кафедри	08.05.2023	Виконано

Дата видачі завдання «23» січня 2023р.

Студент _____ Рамазанов Р.Ш.
(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Мазурова О.О.
(підпис) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 90с., 30 рис., 5 табл., 32 джерел.

ТЕХНОЛОГІЯ ДОСТУПУ, WEB-САЙТ, ADO.NET, ASP.NET CORE, C#, DAPPER, ENTITY FRAMEWORK, MICROSOFT SQL MANAGEMENT STUDIO, MSSQL, ORM, SQL, VISUAL STUDIO 2022

Об’єктом дослідження є технології доступу до реляційних баз даних та порівняння їх продуктивності з використанням ADO.NET.

Метою роботи є дослідження відмінності у використанні різних технологій доступу до баз даних, в тому числі різних ORM та ADO.NET.

Методи розробки базуються на технологіях Entity Framework, Dapper та ADO.NET з використанням СУБД MS SQL і Web-серверу ASP.NET Core.

У результаті роботи виконано експериментальне дослідження технологій доступу до реляційних баз даних, таких як ENTITY FRAMEWORK, DAPPER та ADO.NET. Розроблено додаток, використовуючи ASP.NET CORE, SWAGGER та мову програмування C#. Спроектовано та створено базу даних MS SQL. Розроблено рекомендації щодо використання досліджуваних технологій доступу до бази даних та експерементально виміряні метрики, за допомогою яких зроблено висновки щодо використання технологій доступу.

ACCESS TECHNOLOGY, ADO.NET, ASP.NET CORE, C#, DAPPER, ENTITY FRAMEWORK, MICROSOFT SQL MANAGEMENT STUDIO, MSSQL, ORM, SQL, VISUAL STUDIO 2022, WEBSITE

The object of the study is access technologies to relational databases and a comparison of their performance using ADO.NET.

The purpose of the work is to study the differences in the use of various database access technologies, including various ORMs and ADO.NET.

Development methods are based on Entity Framework, Dapper and ADO.NET technologies using MS SQL DBMS and ASP.NET Core Web server.

As a result of the work, an experimental study of access technologies to relational databases, such as ENTITY FRAMEWORK, DAPPER and ADO.NET, was performed. Developed application using ASP.NET CORE, SWAGGER and C# programming language. Designed and created MS SQL database. Recommendations for the use of the researched access technologies to the database and experimentally measured metrics were developed, with the help of which conclusions about the use of access technologies were made.

Умови публікації пояснювальної записки

Я, Рамазанов Расул Шамільович, студент гр. ІПЗм-21-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження технологй доступу до реляційних баз даних», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасному світі кожного дня розробляються сотні великих проектів, та з’являються тисячі стартапів із найрізноманітнішими тематиками, розробники починають робити свої проекти, компанії модернезують старі проекти. У кожного з цих видів діяльності є дещо спільне – те, як вони зберігають дані та маніпулюють з ними. Майже всі сучасні проекти мають базу даних, бо взаємодіяти з клієнтом без неї стає просто неможливо. Зараз, яким би твій додаток, сервіс, гра чи будь що інше не було, необхідно щоб у користувача була можливість мати свою облікову сторінку, зберігати прогрес, будь що, що треба зберегти локально або в хмарі, або на сервері. І сучасний розробник має змогу використовувати базу даних напряму або через спеціальні технології доступу до баз даних, в тому числі і через ORM або Micro ORM.

Метою роботи є дослідження відмінностей у використанні різних технологій доступу до баз даних на основі їх експериментального порівняння з замірами відповідних метрик оцінки продуктивності.

У результаті роботи виконано дослідження з відмінностей використання ORM, таких як ENTITY FRAMEWORK та DAPPER, і продуктивності у порівнянні з ADO.NET. Розроблено додаток, використовуючи ASP.NET CORE та мову програмування C#. Спроектовано та створено базу даних MS SQL.

Під час виконання кваліфікаційної роботи був проведений аналіз проблемної області використання технологій доступу з реляційними базами даних на основі публікацій світових та вітчизняних фахівців в проблемній області (див. додаток А). Були розглянуті основні технології доступу до реляційних баз даних на платформі .NET, а також обрана СУБД для проведення дослідження.

Було спроектовано схему та структуру бази даних, створені запити до цієї бази даних, сформовано та описано процес проведення експерименту дослідження. Сформовані метрики для порівняння ефективності використання досліджуваних технологій доступу.

Робота пройшла успішну перевірку на академічну доброчесність (див. додаток Б).

На основі плану проведення дослідження та його результатів було розроблено презентацію (див. додаток В). За результатами роботи були створені тези доповіді на двадцять сьомий міжнародний молодіжний форум «РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ В XXI ст.» , а також, підготовлено до подачі наукову статтю «Дослідження технологій доступу до реляційних баз даних» у науковому журналі «Біоніка інтелекту» (див. додаток Г).

Також, робота перевірена на відповідність вимогам оформлення (див. додаток Д).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ВИКОРИСТАННЯ ТЕХНОЛОГІЙ ДОСТУПУ ДО БАЗ ДАНИХ

SQL має безліч переваг, завдяки чому вона є популярною та дуже вимаганою. Ця мова є надійною та ефективною у взаємодії з базами даних [13]. Деякі з переваг SQL включають наступне:

- швидша обробка запитів;
- не потребує навичок кодування;
- стандартизована мова;
- портативність;
- інтерактивна мова.

SQL здатна швидко та ефективно обробляти великі обсяги даних. Операції, такі як вставка, видалення та маніпулювання даними, також здійснюються майже миттєво [7].

SQL пропонує зручну синтаксичну конструкцію, яка дозволяє здійснювати пошук даних без великої кількості коду. Всі основні ключові слова, такі як SELECT, INSERT INTO, UPDATE та інші, доступні для використання. Синтаксичні правила в SQL не є складними, що робить його досить зручною мовою. Завдяки документації та тривалому розвитку протягом багатьох років, SQL надає єдину платформу для користувачів по всьому світу.

SQL – проста мова для вивчення та розуміння. Навіть складні запити можна виконувати швидко, отримуючи відповіді всього за кілька секунд.

Серед недоліків SQL можна виділити наступне:

- комплексний інтерфейс. Для роботи з SQL необхідно використовувати спеціальні програми або інтерфейси;
- вартість. Деякі продукти або системи управління базами даних, які підтримують SQL, можуть бути дорогими для придбання або ліцензування ;
- частковий контроль. SQL не забезпечує повний контроль над безпекою та цілісністю даних.

Застосування SQL:

- SQL використовується розробниками та DBA (адміністраторами баз даних) для написання сценаріїв інтеграції даних;
- він використовується для роботи з аналітичними запитами для аналізу даних і отримання з них інстинктів;
- отримання інформації;
- модифікація/маніпулювання даними та таблицею бази даних, наприклад вставка, видалення та оновлення.

Система управління базами даних (СУБД) є комплексом програмних та апаратних засобів, що дозволяють проектувати, налаштовувати та адмініструвати бази даних (БД). СУБД забезпечує безпеку, цілісність та надійність зберігання даних, а також надає можливість керувати доступом до адміністрування БД [11].

СУБД забезпечує:

- взаємодія з інформацією, що зберігається на зовнішніх накопичувачах;
- роботу з «гарячими» даними, які розташовані в оперативній пам'яті або SSD;
- логування кожного етапу під час роботи з БД;
- підтримку більшості форматів БД.

СУБД складається з:

- ядра; підтримує звітність, відповідає за управління даними у ОЗП та на зовнішніх накопичувачах;
- процесор мови БД; обробляє вхідні команди користувача або адміністратора. Оптимізує запити на створення та модифікування даних;
- підсистеми підтримки часу виконання. Дозволяє інтерпретувати програмні рішення з поточними базами даних, а також створювати інтерфейси взаємодії (API) з СУБД;
- допоміжного ПЗ - системні утиліти, які надають додаткові функції з менеджменту та адміністрування баз даних.

СУБД класифікуються на різні типи залежно від моделей використовуваних даних, способів надання доступу до БД, і навіть за рівнем розподіленості [1].

Залежно від моделі даних СУБД бувають:

– мережевими;
– ієрархічними;
– реляційними;
– об'єктно-реляційними;
– об'єктно-орієнтованими та іншими.

Відповідно до методу надання доступу до БД СУБД поділяються на:

– вбудовані;
– "Клієнт-сервер";
– "Файл-сервер".

За рівнем розподіленості СУБД бувають:

– гео-розподіленими (складові елементи єдиної СУБД рознесені за різними обладнаннями, які можуть бути географічно розташовані в різних місцях);
– локальними (ПЗ встановлено в одному ЦОДі).

Microsoft SQL Server – це популярна система управління базами даних (СУБД), розроблена корпорацією Microsoft. Ця програма призначена для зберігання та обробки даних. Взаємодія з MS SQL Server дозволяє користувачам надсилати запити та отримувати результати, як локально, так і через мережу. Робота програми полягає в тому, що вона відкриває мережевий порт, приймає команди та повертає результат [2].

База даних представляє собою структурований набір відомостей про об'єкти. Зазвичай, для управління базами даних використовуються спеціальні програмні засоби, такі як системи управління базами даних (СУБД) [14].

В залежності від типу бази даних, її логічна структура може мати різний опис. Ця різниця визначає, яку саме базу даних використовують у розробці конкретного продукту або технології.

Реляційні бази даних вважаються найстарішим типом баз даних, з теоретичними основами, запропонованими британським вченим Едгаром Коддом у 1970 році [15]. У реляційних базах даних дані організовані у вигляді таблиць з рядками та стовпцями. Рядки представляють відомості про конкретні об'єкти

(значення властивостей), а стовпці представляють самі властивості об'єктів, відомості про які зберігаються у полях [4].

Переваги та недоліки реляційних баз даних дійсно випливають з їх жорсткої структуризації та типізації відомостей про об'єкти. З одного боку, це дозволяє оптимізувати зберігання та індексування даних шляхом використання нормалізації або денормалізації. Це дає можливість ефективно виконувати запити та забезпечувати цілісність даних. З іншого боку, реляційні бази даних можуть виявитись складними у випадку зберігання та обробки погано структурованих даних, наприклад, об'єктів кешування, або зовсім неструктурованих даних, які походять з різних джерел. В таких випадках можуть виникати проблеми з проектуванням та організацією бази даних, а також з виконанням складних запитів.

Для боротьби з цими обмеженнями розробили сімейство нереляційних БД.

Нереляційна база даних – це база даних, у якій на відміну більшості традиційних систем баз даних не використовується таблична схема рядків і стовпців. У цих базах даних застосовується модель зберігання, оптимізована під конкретні вимоги типу даних, що зберігаються [16].

Технології доступу до даних є прошарком між API конкретного сервера та програмою користувача, надаючи програмісту простий уніфікований механізм роботи з даними.

На сьогоднішній день існує безліч технологій доступу до даних, таких як BDE, OLE, ODBC, DAO, ADO та інші, і досі розробляються нові, надійніші, зручніші в роботі та швидкодіючі технології [3].

Механізми доступу до баз даних знижують складність обміну інформацією з базами, проте інтерпретація результатів їхньої роботи також досить трудомістка. Тому реалізовані набори компонентів, призначені взаємодії з механізмами обміну.

ADO (ActiveX Data Objects) – це технологія доступу до даних, розроблена Microsoft. Вона є надбудовою над механізмом доступу OLE DB і призначена для уніфікації роботи з постачальниками даних OLE DB. ADO забезпечує зручний та надійний доступ до даних, хоча деякі операції можуть бути трохи повільнішими, ніж у технологіях BDE (Borland Database Engine) та dbExpress.

ADO добре підходить для роботи з системами управління базами даних (СУБД) від Microsoft, такими як MS Access і MS SQL Server. Одна з переваг ADO полягає в тому, що вона не потребує додаткових компонентів (бібліотек або драйверів), оскільки вони вже присутні на комп'ютері користувача [17].

Механізм ADO надає кілька основних COM-об'єктів, що використовуються для отримання та управління інформацією (є додаткові COM-об'єкти, що розширюють функціональність ADO):

- Connection для керування з'єднанням з базою даних та надсилання запитів постачальнику даних;
- Command для керування інформацією про запит до бази даних або команди;
- Recordset, що містить таблицю, яка є результатом запиту до бази даних;
- Field, що містить опис поля у таблиці, повернутий постачальником даних. Список всіх полів таблиці міститься у подоб'єкті Fields об'єкта RecordSet;
- Error, що містить розширену інформацію про помилку, про яку повідомив постачальник даних. Якщо кілька помилок, доступ до них можна отримати за допомогою об'єкта Errors.

Для проведення дослідження було обрано СУБД Microsoft SQL Server та відповідно база даних SQL, бо вони є дуже популярними серед розробників на платформі .NET, також з цією СУБД та супутньою базою даних є досвід праці як із лабораторними, практичними, курсовими роботами, так і комерційний досвід праці, що надає перевагу при проектуванні бази даних та розробці програмного застосунку, який буде використовувати дану СУБД та БД.

ORM (Object-Relational Mapping, об'єктно-реляційне відображення) – це підхід до програмування, який дозволяє створювати "віртуальну об'єктну базу даних". ORM технологія забезпечує зв'язок між об'єктно-орієнтованим програмуванням і реляційними базами даних, дозволяючи розробникам працювати з даними у вигляді об'єктів, замість написання прямих SQL-запитів до бази даних.

Завдяки цій технології, розробники можуть використовувати свою улюблену мову програмування для роботи з базою даних замість того, щоб писати прямі SQL-

запити або збережені процедури. Це дозволяє прискорити процес розробки програм, особливо на початкових етапах, оскільки розробники можуть сконцентруватись на логіці програми, використовуючи знайомі мови програмування та звичні парадигми.

При використанні ORM слід пам’ятати також і про іхні недолікі:

- ефективність виконання програми та режим фіксованого мислення приносяться в жертву, що знижує гнучкість розробки;
- з точки зору структури системи, системи, що використовують ORM, зазвичай являють собою багатошарові системи, і чим більша система, тим нижча ефективність. ORM – це повністю об'єктно-орієнтований підхід, і об'єктно-орієнтований підхід також вплине на продуктивність;
- при розробці системи, зазвичай виникають проблеми із продуктивністю. Проблеми з продуктивністю здебільшого викликані неправильними алгоритмами та неправильним використанням баз даних. Код, згенерований ORM, зазвичай, навряд чи напише дуже ефективний алгоритм, і він, швидше за все, неправильно використовуватиметься у додатках баз даних. Це в основному відображається у витягуванні постійних об'єктів та обробці даних. Якщо використовується ORM, програміст, ймовірно, витягне всі дані в об'єкт пам'яті, а потім відфільтрує і обробить їх, що є причиною проблем з продуктивністю;
- при забезпеченні сталості об'єктів ORM зазвичай зберігає всі властивості, що іноді небажано [5].

2 ПОСТАНВКА ЗАДАЧІ

Метою кваліфікаційної роботи є дослідження існуючих технологій доступу до баз даних, таких як ORM, Micro ORM та порівняння їх ефективності, часу, вартості та інших властивостей з ADO.NET.

Об’єктами дослідження виступають підходи доступу до баз даних та особливості їх використання у контексті реляційних баз даних на платформі .NET та СУБД MS SQL.

Таким чином, під час дослідження необхідно вирішити наступні завдання:

- провести аналіз технологій доступу до реляційних баз даних для обраної СУБД та обрати найкращі для дослідження;
- провести планування експериментального дослідження (обрати предметну область та розробити схему БД, обрати метрики для оцінки результатів експерименті) ;
- реалізувати програмні рішення під обрані технології доступу до бази даних;
- провести експерименти та сформулювати рекомендації щодо використання технологій доступу до БД.

3 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Аналіз існуючих технологій доступу до реляційних баз даних

Для доступу до реляційних баз даних в середовищі .NET існує багато технологій, серед яких можна виділити ті, які будуть розглядатися в даній роботі:

- ADO.NET (ActiveX Data Objects .NET) – це фреймворк, що входить до складу .NET Framework і дозволяє взаємодіяти з реляційними базами даних, такими як Microsoft SQL Server, Oracle, MySQL, і багатьма іншими. ADO.NET включає класи та компоненти для роботи з даними, такі як DataReader, DataSet, SqlConnection, SqlDataAdapter і багато інших;
- ORM (Object-Relational Mapping) – це технологія, яка дозволяє взаємодіяти з реляційними базами даних через об'єкти класів, замість написання SQL-запитів. До найпопулярніших ORM фреймворків для .NET належать Entity Framework (EF), NHibernate, LINQ to SQL та багато інших;
- Micro-ORM – це легковажніша альтернатива повноцінному ORM фреймворку, яка дозволяє використовувати прямий SQL для взаємодії з базою даних, при цьому забезпечуючи зручне мапування результатів запиту на об'єкти .NET. До найпопулярніших Micro-ORM для .NET можна віднести Dapper, OrmLite, PetaPoco та багато інших [19].

Кожна з цих технологій має свої переваги та недоліки, тому вибір між ними залежить від потреб проекту та вимог до продуктивності, безпеки, зручності розробки та інших факторів.

ORM вибудовує моделі об'єктно-орієнтованої програми з високим рівнем абстракції. Інакше кажучи, вона створює рівень логіки без основних деталей коду. Відображення описує відносини між об'єктом та даними, не знаючи, як ці дані структуровані. Далі модель можна використовувати для підключення додатка до SQL, який необхідний для керування операціями з даними. Цей тип коду не потрібно переписувати, що заощаджує час розробки.

Отже, було описане загальне поняття ORM-систем, принципом їхньої роботи, тепер розглянемо деякі з них докладніше.

Entity Framework (EF) є технологією доступу до даних, яка працює на платформі .NET Framework від Microsoft і забезпечує об'єктно-орієнтоване відображення даних в базі даних (ORM). Він дозволяє взаємодіяти з даними за допомогою LINQ (Language Integrated Query) як LINQ to Entities, а також за допомогою Entity SQL.

Entity Framework також надає підтримку для побудови web-рішень за допомогою ADO.NET Data Services. Він інтегрується з Windows Communication Foundation (WCF) та Windows Presentation Foundation (WPF), що дозволяє створювати багаторівневі програми, реалізуючи шаблони проектування, такі як MVC (Model-View-Controller), MVP (Model-View-Presenter) або MVVM (Model-View-ViewModel) [20].

NHibernate - ще одна з найбільш широко використовуваних ORM для .NET, яка спрощує завдання, пов'язані зі збереженням даних. Це не найвдаліше рішення для додатків, які використовують лише процедури, що зберігаються для реалізації бізнес-логіки в БД. Однак вона найбільш корисна для об'єктно-орієнтованих моделей предметної галузі та бізнес-логіки у .NET середнього рівня. NHibernate може допомогти видалити або інкапсулювати SQL код конкретного постачальника, а також виконати завдання перетворення набору результатів з табличного подання в графік об'єктів [21].

Dapper – ця мікро-ORM набула великої популярності і навіть була включена як опція ORM у Visual Studio при створенні нового проекту. Однією з її переваг є функціональність мапера, яка багато в чому схожа на LINQ в ядрі EF, а також надає методи розширення, що спрощують відправлення операторів T-SQL до бази даних [22].

DevExpress XPO – це інструмент об'єктно-реляційного зіставлення, який опрацьовує всі аспекти створення бази даних та збереження об'єктів, дозволяючи зосередитися на бізнес-логіці програми, а не на складності БД. XPO ORM Library пропонує перші кроки розробки Code First, Model First та Database First [23].

DataObjects.NET – платформа, що поєднує у собі вбудовувану базу даних, засоби реалізації бізнес-логіки і готовий рівень доступу до даних (ORM), підтримує

як найпоширеніші БД (Microsoft SQL Server, Oracle, PostgreSQL), так і вбудовану базу даних [24]. Використання бібліотеки дозволяє суттєво скоротити час розробки додатків, що працюють з реляційними даними – бібліотека бере на себе практично всі функції, пов'язані із взаємодією з сервером БД, виконуючи їх прозоро (тобто не вимагаючи написання коду, який забезпечує виконання) для розробника. Серед її унікальних особливостей – автоматичне оновлення схеми БД, найбільш повна підтримка успадкування (наприклад, можливі запити на інтерфейс, що підтримується), вбудований механізм повнотекстового індексування та пошуку, управління правами доступу до об'єктів [6].

Тепер можна розглянути задачу вибору ORM та Micro-ORM, які будуть порівнюватись з ADO.NET. Для початку треба сформувати список з ORM та Micro-ORM, які будуть розглянуті і проаналізовані, а саме:

– Entity Framework;
– Dapper;
– Nhibernate ;
– DevExpress XPO;
– DataObjects.NET.

Тепер необхідно сформувати за якими критеріями обрані ORM будуть досліджуватись

– підхід процесу розробки – як буде створюватися база даних чи її реалізація у застосунку;
– доступність матеріалів – кількість відкритих ресурсів для набуття досвіду та навичок використання данної ORM;
– продуктивність – оцінка (час), при спробі прочитати 500 тис. записів з таблиці, даний експеримент наведено за посиланнями: ORM read-performance та http://www.ormeter.net/;
– складність у використанні – час, необхідний для вивчення та початку використання данної ORM;
– популярність (GitHub) – кількість зроблених fork у GitHub для данної ORM, що свідчать зацікавленності людей у ній.

Існує три основні підходи до моделювання сутностей: спочатку код (Code First), спочатку модель (Model First) і спочатку база даних (Database First).

Підхід Code First допомагає вам створювати сутності у вашій програмі, зосереджуючись на вимогах домену. По суті, за допомогою цього підходу ви можете дотримуватися дизайну, керованого доменом (DDD). Після того, як ваші сутності та конфігурації визначено, ви можете створити базу даних на льоту. Підхід Code First дає вам більше контролю над своїм кодом – вам більше не потрібно працювати з автоматично створеним кодом [27].

Ви можете використовувати підхід Database First, якщо база даних уже розроблена та готова. У цьому підході модель даних сутності створюється з основної бази даних [28].

У підході Model First ви можете спочатку створити моделі даних сутності, а потім з них створити базу даних. Зазвичай ви створюєте моделі, визначаєте сутності та їхні зв’язки, а потім створюєте базу даних із цієї визначеної моделі [29].

Після формування критеріїв необхідно сформувати шкалу для оцінювання тих критеріїв, які мають нетривіальні значення, такі як складність використання та інші.

Формування шкали оцінок за критеріями:

а) підхід до процесу розробки:
   1) один підхід – 1;
   2) два підходу – 5;
   3) три підходу – 10.

б) складність у використанні:
   1) висока – 1;
   2) середня – 5;
   3) низька – 10.

в) доступність матеріалів:
   1) низька – 1;
   2) середня – 5;
   3) висока – 10.

Наступним кроком буде моделювання таблиці з описаними альтернативами, критеріями та заповненими оцінками (див. табл. 3.1).

Таблиця 3.1 – Проведення оцінки для альтернатив

	Entity Framework	Dapper	Nhibernate	DevExpress XPO	DataObjects. NET
Підхід процесу розробки	Code First, Model First, Database First	Database First	Code First, Model First, Database First	Code First, Model First, Database First	Code First, Model First, Database First
Складність у використанні	Низька	Середня	Висока	Середня	Середня
Популярність (GitHub)	2.9k forks	3.6k forks	915 forks	84 forks	18 forks
Продуктивність	1700 мс	650 мс	850 мс	1200 мс	1100 мс
Доступність матеріалів	Висока	Висока	Середня	Низька	Низька

Тепер необхідно привести значення у таблиці до єдиного вигляду, для цього необхідно зробити нормування значень, далі будуть описані види нормувань та по яким полям буде виконуватись нормування.

Minmax буде використовуватись для популярності. Для розрахунку буде використовуватись ця формула 3.1:

$$f = \frac{(f\text{значення} - fmin)}{(fmax - fmin)} \tag{3.1}$$

Обернене еталоне нормування буде робитись для продуктивності, бо в даному випадку чим меньше час обробки для читання даних, тим краще. Тоді формула буде виглядати (формула 3.2):

$$f = \frac{f\text{еталон}}{f\text{значення}} \tag{3.2}$$

Після нормування і приведення кожного зі значень до вигляду, де воно є у проміжку від 0 до 1 (див. табл. 3.2), слід розставити для кожного критерія вагові коефіцієнти.

Таблиця 3.2 – Проведення нормування

Нормування по minmax та еталону					
	Entity Framework	Dapper	Nhibernate	DevExpress XPO	DataObjects. NET
Підхід процесу розробки	1	0.1	1	1	1
Складність використання	1	0.5	0.1	0.5	0.5
Популярність (GitHub)	0.80	1	0.25	0.018	0
Продуктивність	0.38	1	0.76	0.619	0.59
Доступність матеріалів	1	1	1	0.1	0.1

Нормовану таблицю можна побачити у таблиці 3.3.

Вагові коефіцієнти – це коефіцієнти, які показують на скільки той чи інший критерій є важливим, тобто вони відображають важливість критерію для данного дослідження, сума усіх вагових коефіцієнтів дорівнює 1.

Таблиця 3.3 – Нормована таблиця

	Entity Framework	Dapper	Nhibernate	DevExpress XPO	DataObjects. NET
Підхід процесу розробки	1	0.1	1	1	1
Складність використання	1	0.5	0.1	0.5	0.5
Популярність (GitHub)	0.80	1	0.25	0.018	0
Продуктивність	0.38	1	0.76	0.619	0.59
Доступність матеріалів	1	1	1	0.1	0.1

Для продуктивності обрано коефіцієнт 0.35, бо продуктивність при воборі ORM для розробки та дослідження має дуже високий пріоритет. Саме продуктивність і буде порівнюватись у даному дослідженню.

Наступним по значенню коефіцієнтом обрано доступність та кількість матеріалів для навчання чи ознайомлення з документацією – 0.3. Це також є великим коефіцієнтом, бо саме від цього і залежить складність для навчання користування цією ORM. Це є також ключовим фактором при виборі ORM, бо кожному розробнику постійно потрібен матеріал, документація, форуми для обговорення та вирішення проблем, навчальні відео.

Для складності використання було обрано коефіцієнт – 0.2. Від складності використання залежить як швидко розробник інтегрує ORM та зможе її використовувати для існуючої бази даних чи для створення бази даних підходами Code First та Model First, для формування запитів, для міграцій.

У популярності ваговий коефіцієнт – 0.1. Для данного дослідження популярність впливає на впізнаваність серед розробників, не є ключовим фактором як продуктивність чи доступність матеріалів для ORM.

Для підходу процесу розробки також виставлено коефіцієнт – 0.05, бо кожна ORM має свої нюанси при використанні, вони мають різну архітектуру та по-різному реалізовані, тому цей показник не є ключовим, він впливає на різноманіття розробки та інтеграції ORM у систему та дослідження.

Результат розподілення кагових коефіцієнтів можна побачити у таблиці 3.4.

Таблиця 3.4 – Лінійна адитивна згортка

Лінійна адитивна згортка з ваговими коефіцієнтами						
	Entity Framework	Dapper	Nhibernate	DevExpress XPO	Data Objects. NET	Вагові коефі-цієнти
Підхід процесу розробки	1	0.1	1	1	1	0.05
Складність використання	1	0.5	0.1	0.5	0.5	0.2
Популярність (GitHub)	0.80	1	0.25	0.018	0	0.1
Продуктивність	0.38	1	0.76	0.619	0.59	0.35
Доступність матеріалів	1	1	1	0.1	0.1	0.3
Z*	0.763	0.855	0.661	0.3985	0.3865	

За принципом Паретто можна спростити цю таблицю та відкинути «зайві» варіанти. Якщо подивитись на елемент DataObjects.NET, то можна побачити, що в нього немає критеріїв, які є кращими, вони або такі самі, або гірші, тому можно його прибрати і не розглядати далі. Так само можно провести оптимізацію і для інших елементів, яле я вирішив, що потрібно зробити адитивну згортку для усіх елементів таблиці.

Оптимізація за Паретто представлена на рисунку 3.1.

Робимо лінійну адитивну згортку з ваговими коефіцієнтами за формулою 3.3:

$$Z^* = \max \sum_{j=1}^{n} a_j \beta_j a_j \tag{3.3}$$

де α_j – нормуруючі множники,

β_j– вагові коефіцієнти, що відображають відносний внесок окремих критеріїв до загального критерію.

		A1	A2	A3	A4	A5
		Entity Framework	Dapper	Nhibernate	DevExpress XPO	DataObjects.NET
B1	Підхід процесу розробки	1	0.1			
B2	Складність використання	1	0.5	0.1	0.5	0.5
B3	Популярність (GitHub)	0.80	1	0.25	0.018	0
B4	Продуктивність	0.38	1	0.76	0.019	0.59
B5	Доступність матеріалів	1	1	1	0.1	0.1

Рисунок 3.1 – Результат скорочення за принципом Паретто

Проаналізувавши коефіцієнти, можна зробити висновок, що Dapper та Entity Framework є найбільш привабливими ORM та Micro-ORM чим інші для дослідження та порівняння з ADO.NET.

3.2 Аналіз обраних технологій

Для дослідження були обрані ORM Entity Framework, micro ORM Dapper та ADO.NET.

ORM або Object Relational Mapping – це метод об’єктно-орієнтованого програмування, який використовується для зіставлення даних між двома несумісними типами. Dapper називають мікро ORM, оскільки він не надає повної функціональності та функцій, як Entity Framework. Dapper надає вам можливість отримувати дані з реляційної бази даних. Він не підтримує підхід до першої підтримки коду та може працювати із запитами й оновленнями через необроблений SQL. Він не може використовуватися для налаштування класів і має набір методів розширення для ADO.NET. Entity Framework – це ORM з відкритим кодом, який доступний як частина .NET framework. Він відповідає за взаємодію між реляційними базами даних і програмами .NET. Це спрощує код програмування, автоматично генеруючи класи моделі для схеми бази даних. Це дозволяє вам спочатку мати базу даних і створювати з неї сутності або, альтернативно, код програми можна ініціювати до проектування бази даних. Таким чином, будь-які внесені вручну зміни до бази даних потрібно оновити або скопіювати на рівень коду, оскільки зміни буде втрачено [9].

Micro ORM – це тип ORM (Object-Relational Mapping), який є легким та простим у використанні. Він надає базові функції для роботи з базою даних, такі як виконання SQL-запитів та зіставлення результатів запитів з об'єктами у додатку. Micro ORM не має такого великого функціоналу, як у повноцінних ORM, але має високу продуктивність і меншу кількість залежностей, що робить його ідеальним для малих та середніх проектів.

Основна ідея Micro ORM полягає в тому, щоб надати програмісту простий та інтуїтивно зрозумілий спосіб роботи з базою даних, не додаючи при цьому надмірної функціональності, яка може призвести до збільшення навантаження на програму. Micro ORM часто використовується в мовах програмування, таких як C# і Java, і може працювати з різними системами керування базами даних, такими як SQL Server, MySQL та PostgreSQL.

Деякі з найбільш популярних Micro ORM включають Dapper, OrmLite і PetaPoco. Ці бібліотеки надають простий та швидкий спосіб роботи з базами даних

і дозволяють розробникам управляти своїми моделями даних безпосередньо з коду програми, мінімізуючи необхідність написання SQL-запитів вручну.

Dapper – популярний простий інструмент відображення об’єктів. Його розроблено в основному для використання в сценаріях, коли ви хочете працювати з даними строго типізованим способом – як бізнес-об’єктами в програмі .NET, але не хочете витрачати години на написання коду для відображення результатів запиту з даних ADO.NET читачів до екземплярів цих об’єктів. Dapper – це проект з відкритим кодом під ліцензією Apache. Він доступний як пакет Nuget і був завантажений понад 16 мільйонів разів.

Dapper відноситься до сімейства інструментів, відомих як мікро-ORM. Ці інструменти виконують лише частину функціональних можливостей повномасштабних Object Relations Mappers, таких як Entity Framework Core. Характеристики залежать від продукту. У наведеній нижче таблиці наведено загальне уявлення про можливості, які ви можете очікувати в мікро ORM порівняно з ORM (див. табл. 3.5).

Вирішуючи, використовувати Dapper чи ні, слід мати на увазі головну причину його існування – продуктивність. Початкові розробники Dapper використовували попередника Entity Framework Core – короткочасний Linq to SQL. Вони виявили, що продуктивність запитів була недостатньою для зростаючого трафіку, який отримував відповідний сайт (Stackoverflow), тому вони написали власну мікро ORM [10].

Таким чином, Dapper є хорошим вибором у сценаріях, коли дані лише для читання часто змінюються та часто запитуються. Це особливо добре в сценаріях без стану (наприклад, Інтернет), де немає потреби зберігати складні графи об’єктів у пам’яті протягом будь-якого часу.

Таблиця 3.5 – Відмінності ORM від Micro ORM

	Micro ORM	ORM
Map queries to objects	+	+

Caching results	-	+
Change tracking	-	+
SQL generation	-	+
Identity management	-	+
Association management	-	+
Lazy loading	-	+
Unit of work support	-	+
Database migrations	-	+

Dapper не перекладає запити, написані мовами .NET, у SQL, як повномасштабна ORM. Тож вам потрібно навчитися писати запити на SQL або попросити когось написати їх за вас.

Dapper не має реальних очікувань щодо схеми вашої бази даних. Він не залежить від конвенцій так само, як Entity Framework Core, тому Dapper також є хорошим вибором, якщо структура бази даних особливо не нормалізована.

Dapper працює з об’єктом ADO.NET IDbConnection, що означає, що він працюватиме з будь-якою системою баз даних, для якої існує провайдер ADO.NET.

Немає причин, чому ви не можете використовувати як ORM, так і мікро ORM в одному проекті.

ADO.NET (ActiveX Data Objects .NET) – це технологія, призначена для роботи з базами даних у платформі .NET. Вона дозволяє розробникам створювати програми, які можуть взаємодіяти з різними джерелами даних, такими як SQL Server, Oracle, MySQL та інші [12].

ADO.NET надає набір класів та інтерфейсів для доступу до даних, які можуть використовуватися для виконання операцій CRUD (create, read, update, delete) з базою даних. Він також надає функціональність для роботи з транзакціями, обробки помилок та управління з'єднаннями з базою даних [34].

ADO.NET має два основні підходи для доступу до даних: Connected та Disconnected. Connected підхід, використовуючи класи Connection, Command, DataReader, надає прямий доступом до базі даних і призначений виконання операцій із базою даних у режимі реального часу. Disconnected підхід, використовуючи класи DataAdapter, DataSet, DataTable, надає можливість

отримання даних із бази даних та роботу з ними в автономному режимі, що зменшує кількість запитів до бази даних та покращує продуктивність програми [31].

ADO.NET є частиною .NET Framework і є однією з найпопулярніших технологій для доступу до даних у платформі .NET.

До .NET 3.5 розробники часто використовували для написання коду ADO.NET або Enterprise Data Access Block для збереження або отримання даних програми з основної бази даних. Раніше відкривали з'єднання з базою даних, створювали DataSet для отримання або надсилання даних до бази даних, перетворювали дані з DataSet в об'єкти .NET або навпаки для застосування бізнес-правил. Це був громіздкий і схильний до помилок процес. Корпорація Майкрософт надала структуру під назвою «Entity Framework» для автоматизації всіх дій, пов'язаних із базою даних, для вашої програми.

Entity Framework є платформою ORM з відкритим кодом, яка підтримується Microsoft і призначена для розробки додатків на платформі .NET. Використовуючи Entity Framework, розробники можуть працювати з даними на основі об'єктів доменних класів, замість прямого взаємодії з таблицями і стовпцями бази даних, де ці дані зберігаються. Це дозволяє розробникам працювати на вищому рівні абстракції, зосереджуючись на моделюванні домену та бізнес-логіки. [8].

Кросплатформенність: EF Core – це кросплатформна структура, яка може працювати на Windows, Linux і Mac.

Моделювання: EF (Entity Framework) створює EDM (Entity Data Model) на основі сутностей POCO (Plain Old CLR Object) із властивостями get/set різних типів даних. Він використовує цю модель, коли надсилає запити або зберігає дані сутності в базовій базі даних.

Запити: EF дозволяє використовувати запити LINQ (C#/VB.NET) для отримання даних із основної бази даних. Постачальник бази даних перекладе ці запити LINQ на мову запитів, специфічну для бази даних (наприклад, SQL для реляційної бази даних). EF також дозволяє нам виконувати необроблені запити SQL безпосередньо до бази даних [30].

Відстеження змін: EF відстежує зміни, які відбулися в екземплярах ваших об’єктів (значення властивостей), які потрібно надіслати до бази даних.

Збереження: EF виконує команди INSERT, UPDATE і DELETE для бази даних на основі змін, які відбулися у ваших сутностях під час виклику методу SaveChanges(). EF також надає асинхронний метод SaveChangesAsync().

Паралелізм: EF використовує Optimistic Concurrency за замовчуванням, щоб захистити зміни від перезапису, внесені іншим користувачем після того, як дані були отримані з бази даних.

Транзакції: EF виконує автоматичне керування транзакціями під час запиту або збереження даних. Він також надає параметри для налаштування керування транзакціями.

Кешування: EF включає перший рівень кешування з коробки. Таким чином, повторні запити повертатимуть дані з кешу замість потрапляння в базу даних.

Вбудовані угоди: EF дотримується угод щодо шаблону програмування конфігурації та включає набір правил за замовчуванням, які автоматично налаштовують модель EF.

Конфігурації: EF дозволяє нам налаштовувати модель EF за допомогою атрибутів анотації даних або Fluent API, щоб замінити умовні угоди.

Міграції: EF надає набір команд міграції, які можна виконати на консолі диспетчера пакетів NuGet або в інтерфейсі командного рядка для створення базової схеми бази даних або керування нею.

Entity Framework, Dapper та ADO.NET – все це технології для роботи з базами даних у платформі .NET. Кожна з них має свої плюси та мінуси, які можуть визначати вибір залежно від вимог проекту та досвіду розробника [32].

Entity Framework має наступні характеристики:

а) позитивні характеристки:

1) Entity Framework дозволяє працювати з базою даних, використовуючи об'єкти, що робить код більш читабельним та легким для розуміння;
2) EF надає потужний ORM, який дозволяє розробникам позбутися написання SQL-запитів;

3) EF спрощує роботу з реляційними даними, дозволяючи легко створювати зв'язок між таблицями у базі даних.

б) мінуси:

1) при роботі з EF може спостерігатися деяке зниження продуктивності через надмірність функцій, а також багатозадачність та великий обсяг коду;
2) помилки при використанні EF можуть бути досить складними у налагодженні.

Dapper:

а) плюси:

1) Dapper є одним із найшвидших ORM для роботи з базою даних;
2) він легко інтегрується в програму і не вимагає великої кількості коду;
3) під час роботи з Dapper можна писати SQL-запити вручну, що дозволяє отримати максимальний контроль над запитами.

б) мінуси:

1) Dapper не надає потужних функцій для роботи з даними, наприклад Entity Framework;
2) у Dapper не передбачено механізму виявлення змін даних, тому вручну потрібно виконувати синхронізацію з базою даних.

ADO.NET:

а) плюси:

1) ADO.NET дозволяє досягти максимальної продуктивності під час роботи з базою даних;
2) він забезпечує можливість створення зв'язку з різними джерелами даних, включаючи SQL Server, Oracle, MySQL та інші;
3) ADO.NET надає різні способи доступу до даних, включаючи Connected та Disconnected.

б) мінуси:

1) при роботі з ADO.NET може знадобитися більше коду, ніж під час використання Entity Framework або Dapper;

2) ADO.NET не має вбудованих функцій ORM, тому розробнику необхідно писати SQL-запити вручну.

В цілому, вибір між EF, Dapper та ADO.NET залежить від конкретного проекту та вимог до нього. Якщо продуктивність є основним фактором, то Dapper або ADO.NET можуть бути кращими.

Однак, якщо проект має складну структуру бази даних та багато зв'язків між таблицями, то Entity Framework може бути зручнішим у використанні, оскільки він дозволяє працювати з даними у вигляді об'єктів, що спрощує розуміння структури бази даних та взаємозв'язків між таблицями. Загалом вибір технології для роботи з базою даних залежить від конкретних вимог проекту та досвіду розробника [18].

3.3 Планування експерименту

Підсумувавши попередні розділи отримаємо, що дослідження буде проводитись, використовуючи:

- СУБД – MS SQL Server;
- БД – реляційна БД;
- технології, що будуть порівнюватись: ORM Entity Framework та Dapper, необроблений ADO.NET;
- мова програмування – C#;
- веб застосунок – ASP.NET Core.

Експерементальна частина повинна дати нам відповіді на такі питання:

- У кого швидкість роботи вище?
- Де є більше стабільності?
- Чи наявні витоки пам'яті?
- Хто є гнучкішим під час створення запитів?

Тому під час проведення експерементальної частини особливу увагу буде приділено саме швидкості виконання запитів для Dapper, Entity Framework та ADO.NET. Після цього буде досліджуватись, як вони впораються з навантаженнями і на скільки вони покажуть себе стабільними, скільки даних буде

втрачено. І останнім пунктом буде продемонстровано на скільки гнучкими є ці ORM порівняно з ADO.NET.

Отже, для проведення експериментів обрано наступні метрики:

- швидкості виконання запитів (мс);
- швидкості виконання запитів (тіки);
- витрачені ресурси (байти).

Тики (ticks) і мілісекунди (milliseconds) – це одиниці виміру часу, але вони відрізняються один від одного. Тікі – це найменша одиниця виміру часу в .NET Framework, і вона дорівнює одній стотисячній (1/10,000) секунди, тобто одній десятій мілісекунди. Тік використовується для представлення точної кількості тиків, що пройшли з початку часу, наприклад, з початку виконання програми [25].

Під час експерименту замірювались такі характеристики як час виконання запиту для SELECT/INSERT/UPDATE/DELETE операцій у таблицях з різною кількістю записів та об’єм зайнятого дискового простору.

Під час дослідження будуть проведені наступні серії експериментів:

- SELECT/INSERT/UPDATE/DELETE для MS SQL Server за допомогою методу доступу ORM Entity Framework;
- SELECT/INSERT/UPDATE/DELETE для MS SQL Server за допомогою методу доступу micro-ORM Dapper;
- SELECT/INSERT/UPDATE/DELETE для MS SQL Server за допомогою методу доступу ADO.NET.

3.4 Аналіз та моделювання предметної області для дослідження

У якості предметної області була обрана область електронної комерціїї.

Електронна комерція (або e-commerce) – це процес продажу та купівлі товарів та послуг через Інтернет. Це може включати онлайн-магазини, інтернет-аукціони, цифрові товари (наприклад, музика і відео), онлайн-бронювання та оплату послуг (наприклад, готелі, квитки на літак і т.д.), електронну біржу і т.д [26].

Процес електронної комерції включає створення веб-сайту або платформи, на якій користувачі можуть переглянути і вибрати товари та послуги, оформити

замовлення, оплатити їх і отримати доставку товарів. Компанії, що займаються електронною комерцією, можуть використовувати різні стратегії маркетингу та продажів, щоб залучити та утримати клієнтів, такі як реклама, знижки, лояльність, відгуки тощо.

Для проведення дослідження предметною областю буде вважатись інтернет магазин одягу. Аналіз предметної області інтернет-магазину одягу може включати такі аспекти:

- конкуренти: необхідно провести аналіз конкурентів, які вже працюють у цій сфері;
- цільова аудиторія: необхідно визначити цільову аудиторію вашого магазину;
- асортименти: необхідно визначити асортимент вашого магазину. Це можуть бути різні типи одягу, такі як сукні, штани, куртки, джинси та ін., а також аксесуари, такі як сумки, шарфи та ювелірні вироби;
- способи просування: необхідно визначити способи просування вашого магазину. Це може включати різні маркетингові кампанії, такі як реклама в соціальних мережах, на пошукових системах, партнерські програми і багато іншого;
- процеси керування: необхідно визначити процеси керування вашим магазином. Це може включати процеси управління запасами, замовлень, доставки і повернення товарів. Важливо мати чіткі процеси, щоб уникнути помилок та переконатися, що клієнти отримують свої замовлення вчасно.
- система керування контентом: необхідно визначити систему керування контентом вашого магазину. Це може включати платформи для створення та управління контентом на сайті, включаючи опис товарів, фотографії та відео. Важливо мати ефективну систему керування контентом, щоб переконатися, що інформація про товари повна, точна та приваблива для клієнтів.
- оплата та доставка: необхідно визначити способи оплати та доставки вашого магазину;

– послуги та підтримка: необхідно визначити, які послуги та підтримка будуть доступні вашим клієнтам;
– аналітика та звітність: необхідно визначити систему аналітики та звітності вашого магазину. Це може включати інструменти для збору та аналізу даних, такі як Google Analytics, а також звіти про продаж, доходи та прибуток;
– законодавчі вимоги: необхідно враховувати законодавчі вимоги щодо інтернет-магазинів одягу.

Загальну діаграму класів можна побачити на рисунку 3.2.

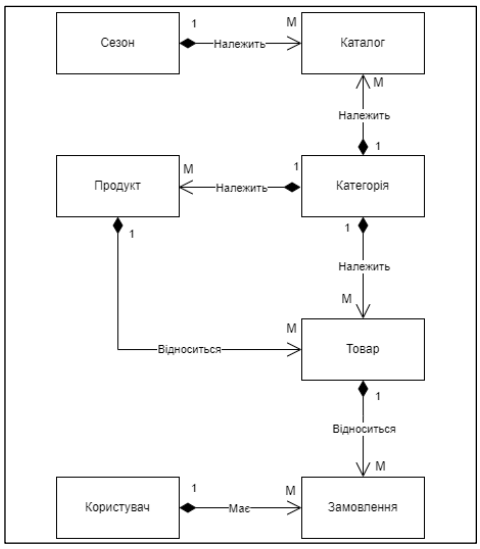


Рисунок 3.2 – Загальна діаграма класів предметної області

E-commerce (електронна комерція) є дуже важливою для інтернет-магазинів одягу з кількох причин.

По-перше, електронна комерція дозволяє інтернет-магазину продавати товари в будь-який час, без обмежень годин роботи і днів тижня. Клієнти можуть зробити замовлення в зручний для них час, що забезпечує зручність інтернет-магазину.

По-друге, електронна комерція дозволяє інтернет-магазину знизити витрати на оренду приміщення, оплату заробітної плати та інші витрати, пов'язані з традиційним магазином. Це дає можливість інтернет-магазину пропонувати товари за більш конкурентоспроможними цінами, що приваблює більше клієнтів.

По-третє, електронна комерція дає можливість інтернет-магазину дуже швидко і ефективно оновлювати асортимент товарів і відслідковувати попит на певні товари. Це дозволяє магазину швидко реагувати на змінні умови ринку і вносити необхідні зміни в своє пропозицію.

По-четверте, електронна комерція дозволяє інтернет-магазину отримати доступ до глобального ринку, привернути більше клієнтів з різних країн і збільшити свої продажі.

Отже, електронна комерція є надзвичайно важливою для інтернет-магазину одягу, оскільки вона дозволяє забезпечити зручність клієнтів, знизити витрати, швидко реагувати на зміни в умовах ринку та отримати доступ до глобального ринку.

3.5 Проектування бази даних для дослідження

Для проведення експериментального дослідження є необхідність проектування та наповнення бази даних, що, у наслідку, буде використовуватись для тестування різних методів доступу до бази даних.

Проектування бази даних для інтернет-магазину одягу включає кілька етапів, які можна описати наступним чином:

- визначення вимог до БД: Необхідно визначити, які дані зберігатимуться у БД, такі як інформація про товари, замовлення, покупці тощо;
- розробка концептуальної моделі: На цьому етапі створюється концептуальна модель БД, яка визначає основні сутності та зв'язки між ними, наприклад, сутності "товар", "категорія товару", "замовлення", "покупець" тощо;
- розробка логічної моделі: На цьому етапі створюється логічна модель БД, яка визначає структуру таблиць, зв'язок між таблицями, ключі та атрибути. Наприклад, таблиці "товари", "замовлення", "покупці", "категорії товарів" тощо;
- розробка фізичної моделі: На цьому етапі визначаються характеристики зберігання даних, такі як типи даних, індекси, розміри таблиць та інші

параметри. Також цьому етапі вибираються СУБД (система управління базами даних) і визначається спосіб зберігання даних;

- реалізація БД: На цьому етапі створюється сама БД із використанням обраної СУБД. Створюються таблиці, індекси та інші об'єкти бази даних;
- тестування: На цьому етапі перевіряється коректність роботи БД, її продуктивність та надійність. Тестування має включати перевірку наявності необхідних функцій та коректність обробки даних, а також тестування на високі навантаження та безпеку;
- експлуатація та супровід: Після впровадження системи в експлуатацію її необхідно супроводжувати та розвивати, а також вносити зміни до БД відповідно до змін у вимогах до системи. Також слід забезпечувати регулярне резервне копіювання даних та моніторинг працездатності БД.

На етапі проектування треба виділити опорні сутності, згідно яким будуть будуватися зв’язки:

- сезон, повинен мати назву та дату початку, описується таблицею «Seasons»;
- каталог, у сезоні можуть бути декілька каталогів різних категорій, описуется таблицею «Catalogs»;
- категорії, мають назву категорії, описуеться таблицею «Categories»;
- продукт, описує сутність продукта, який може бути використаний, має опис головних критеріїв, таких як, назва, ціна, кольор, опис, категорія, описується таблицею «Products»;
- товар, мають посилання на продукт та залежать від каталогу, який буде використовуватись у сезоні, описується таблицею «Goods»;
- замовлення, має посилання на користувача, який зробив замовлення, а також посилання на товар, описується таблицею «Orders»;
- користувач, описує сутність користувача, має такі поля як ім’я, пошта, телефон, описується таблицею «Users».

На рисунку 3.3 приведена схема бази даних.

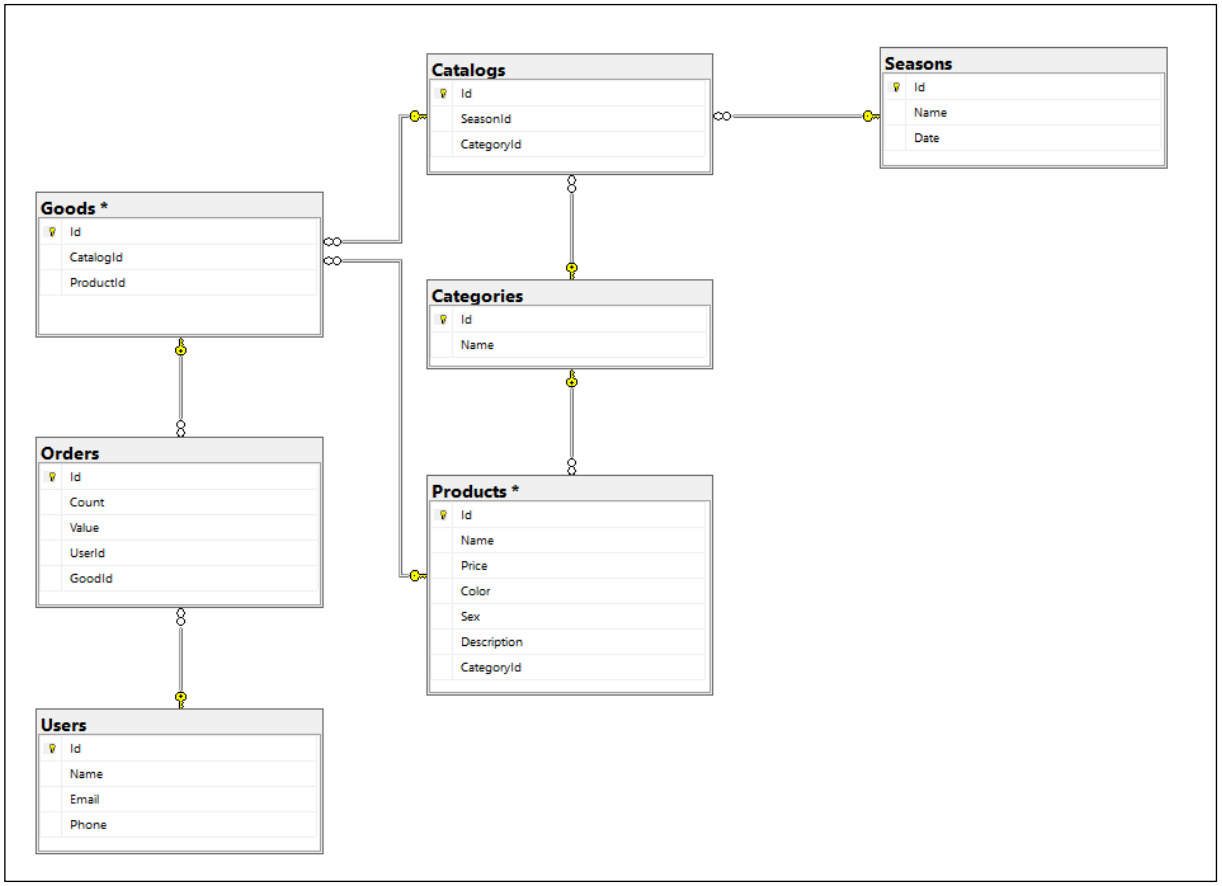


Рисунок 3.3 – Схема бази даних

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Опис створення бази даних для роботи з методами доступів

Необхідно розробити систему, яка зможе підтримувати підлкючення до бази даних через обрані ORM та ADO.NET та навантажити сервер бази даних. Розглянемо підхід Code First для Entity Framework. Тому була розроблена діаграма класів, яка наведена на рисунку 4.1.

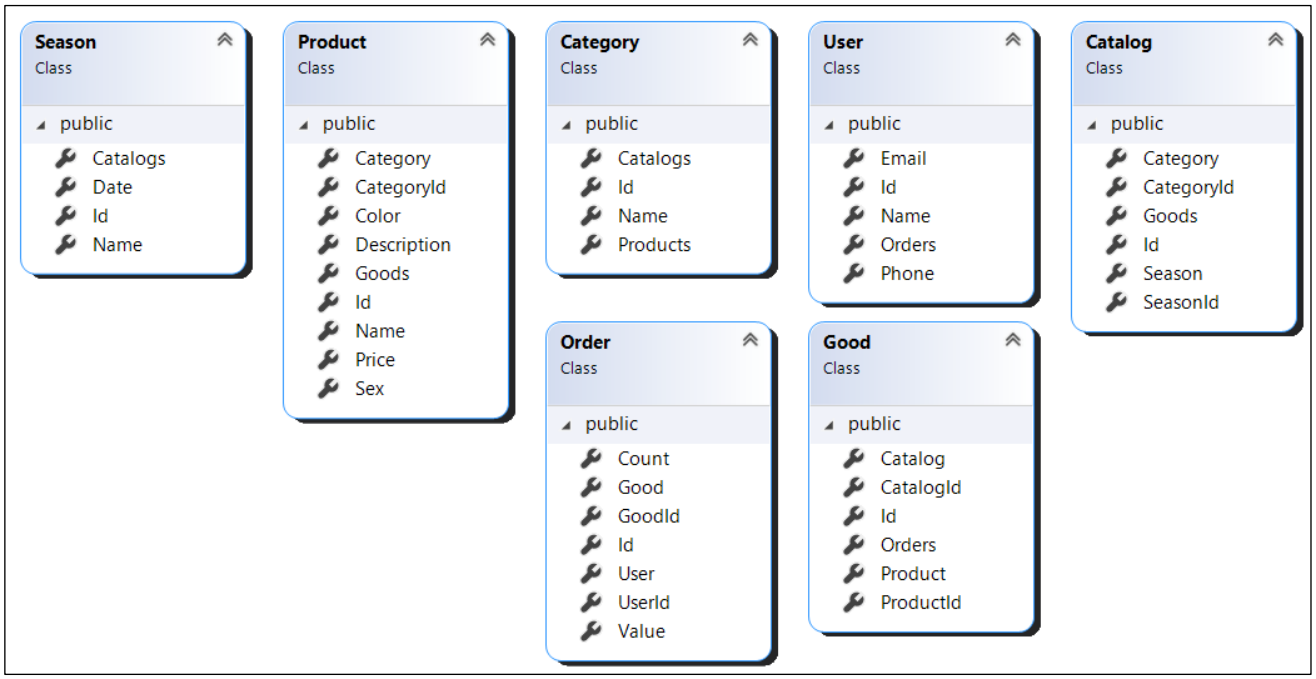


Рисунок 4.1 – Діаграма класів

На діаграмі можна побачити 7 класів: Season, Product, Category, User, Catalog, Order, Good.

Клас User – це модель користувача, в якого є аккаунт, і який може робити замовлення на сайті. У цій моделі є такі поля, як Email, Id, Name, Orders, Phone. Де поле Id – є унікальним ключем для моделі. Поле Orders є колекцією замовлень для створення зв’язку один до множини через Entity Framework Code First. Поля Name, Phone та Email є строковими даними. Структуру класа можна побачити на рисунку 4.2.

Клас Season – це модель сезону, в якому можуть бути ті чи інші каталоги одягу зі своїм асортиментом товарів, у кожного сезону є своє ім’я та дата по якій його легко знайти, бо кожен сезон має свою дату початку.

```
public class User
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Email { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Phone { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public List<Order> Orders { get; set; } = new List<Order>();
}
```

Рисунок 4.2 – Клас User

У цій моделі є такі поля, як Id, Name, Catalogs та Date. Де поле Id – є унікальним ключем для моделі. Поле Name є строковими даними. Поле Date представляю собою дату початку сезону. Поле Catalogs є колекцією каталогів для створення зв’язку один до множини через Entity Framework Code First. Структуру класа можна побачити на рисунку 4.3.

```
public class Season
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public DateTime Date { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public List<Catalog> Catalogs { get; set; } = new List<Catalog>();
}
```

Рисунок 4.3 – Клас Season

Клас Catalog – це модель каталогу для одягу, в якому знаходяться товари відповідної категорії, каталог має сезон, до якого відноситься. У цій моделі є такі поля, як Id, SeasonId, Goods та CategoryId, а також поля Season та Category. Де поле Id – є унікальним ключем для моделі. Поле SeasonId є зовнішнім ключем до таблиці сезонів. Поле CategoryId є зовнішнім ключем до таблиці категорій. Поле Goods є

колекцією товарів у каталозі для створення зв’язку один до множини через Entity Framework Code First. А поля Season та Category створюють зв’язок множина до одного через Entity Framework Code First. Структуру класа можна побачити на рисунку 4.4.

```
public class Catalog
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int SeasonId { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public Season Season { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int CategoryId { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public Category Category { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public List<Good> Goods { get; set; } = new List<Good>();
}
```

Рисунок 4.4 – Клас Catalog

Клас Category – це модель категорії для одягу, в якій є назва категорії для каталогів і продуктів. У цій моделі є такі поля, як Id, Catalogs, Products та Name. Де поле Id – є унікальним ключем для моделі. Поле Name є строкою для імені категорії. Поля Catalogs та Products є колекціями каталогів та продуктів у каталозі для створення зв’язку один до множини через Entity Framework Code First. Структуру класа можна побачити на рисунку 4.5.

```
public class Category
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public List<Catalog> Catalogs { get; set; } = new List<Catalog>();

    0 references | 0 changes | 0 authors, 0 changes
    public List<Product> Products { get; set; } = new List<Product>();
}
```

Рисунок 4.5 – Клас Category

Клас Product – це модель одягу, в якій міститься інформація щодо данного продукту. У цій моделі є такі поля, як Id, Name, Price, Color, Sex, Description, CategoryId, Category та Goods. Де поле Id – є унікальним ключем для моделі. Поле Name є строкою для назви продукту. Поле Price для запису ціни продукта. Поле Color є строкою для опису кольору продукту. Поле Sex є строковим полем для визначеня полу продукту. Поле Description є строковим для опису продукту. Поле CategoryId є зовнішнім ключем для таблиці Category. Поле Goods є колекцією товарів продуктів для створення зв'язку один до множини через Entity Framework Code First. Поле Category створює зв'язок множина до одного через Entity Framework Code First. Структуру класа можна побачити на рисунку 4.6.

```
public class Product
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public decimal Price { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Color { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Sex { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public string Description { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int CategoryId { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public Category Category { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public List<Good> Goods { get; set; } = new List<Good>();
}
```

Рисунок 4.6 – Клас Product

Клас Goods – це модель товару одягу, який вже знаходиться в каталозі і який вже може бути доданий у замовеня користувачем. У цій моделі є такі поля, як Id, CatalogId, Catalog, ProductId, Product та Orders. Де поле Id – є унікальним ключем

для моделі. Поля CatalogId та ProductId є зовнішніми ключами для таблиць Catalog і Product. Поле Orders є колекцією замовлень товару для створення зв'язку один до множини через Entity Framework Code First. А поля Catalog та Product створюють зв'язок множина до одного через Entity Framework Code First Структуру класа можна побачити на рисунку 4.7.

```
public class Good
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int CatalogId { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public Catalog Catalog { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int ProductId { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public Product Product { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public List<Order> Orders { get; set; } = new List<Order>();
}
```

Рисунок 4.7 – Клас Good

Клас Order – це модель замовлення товару. У цій моделі є такі поля, як Id, Count, Value, UserId, User, GoodId та Good. Де поле Id – є унікальним ключем для моделі. Поле Count є чисельним і показує кількість замовленного товару. Поле Value відображає ціну замовлення. Поля UserId та GoodId є зовнішніми ключами для таблиць User і Good. А поля User та Good створюють зв'язок множина до одного через Entity Framework Code First Структуру класа можна побачити на рисунку 4.8.

```
public class Order
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int Count { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public decimal Value { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int UserId { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public User? User { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public int GoodId { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public Good? Good { get; set; }
}
```

Рисунок 4.8 – Клас Order

4.2 Реалізація програмного застосунку для проведення досліджень

Для проведення дослідження необхідно розробити систему, яка зможе показати кінцевий результат експериментальної частини. Для успішного проведення експерименту необхідно буде використовувати декілька ORM та ADO.NET паралельно, або послідновно, тобто мати декілька підключень до бази даних і щоб не переключати для кожного різного способу доступу до бази даних вручну, необхідно буде реалізувати у системі механізм автопідключення, залежно від методу доступу до бази даних. Також, щоб провести експеримент необхідно надмірно навантажити методи доступа до бази даних, тобто мати велику кількість даних та засоби, щоб перетворювати дані з таблиць у інші види, та мати змогу скористатися складними або комплексними запитами до бази даних.

Для дослідження був розроблений веб додаток, який має змогу маніпулювати даними бази даних, надсилати запити та вимірювати час виконання запитів та інших метрик для різних методів доступу до бази даних.

Для розробки веб застосунку було використано ASP.NET Core Web API.

ASP.NET Core Web API – це фреймворк для розробки веб-сервісів, що базуються на архітектурі RESTful і призначений для роботи в середовищі .NET Core. Ось основні компоненти архітектури ASP.NET Core Web API:

- маршрутизація (Routing): ASP.NET Core Web API використовує систему маршрутизації для зв'язування URL-адрес з методами контролерів. Маршрутизація визначається класом Startup.cs і може бути налаштована за допомогою атрибутів маршрутизації на методах контролерів;
- контролери (Controllers): Контролери є класами, які обробляють запити від клієнтів. Вони містять методи, які можуть виконувати певні дії та повертати дані у форматі JSON, XML тощо;
- моделі (Models): Моделі використовуються для представлення даних, що передаються між клієнтом та сервером. Вони можуть бути простими класами з властивостями, а також містити логіку валідації даних;

– фільтри: Фільтри дозволяють виконувати дії перед або після виконання методів контролерів. Це може бути корисним, наприклад, для перевірки автентифікації користувача або для логування запитів;
– middleware: Middleware – це компоненти, які обробляють запити та відповіді. ASP.NET Core Web API включає кілька вбудованих Middleware, наприклад, для обробки винятків, авторизації та кешування;
– серіалізація даних (Data serialization): ASP.NET Core Web API використовує формати серіалізації даних, такі як JSON або XML, для передачі даних між клієнтом та сервером.

В цілому, архітектура ASP.NET Core Web API дозволяє розробляти швидкі, масштабовані та надійні веб-сервіси, які можуть бути інтегровані у різні програми та платформи. Приклад архітектури ASP.NET Core Web API можна побачити на рисунку 4.9.

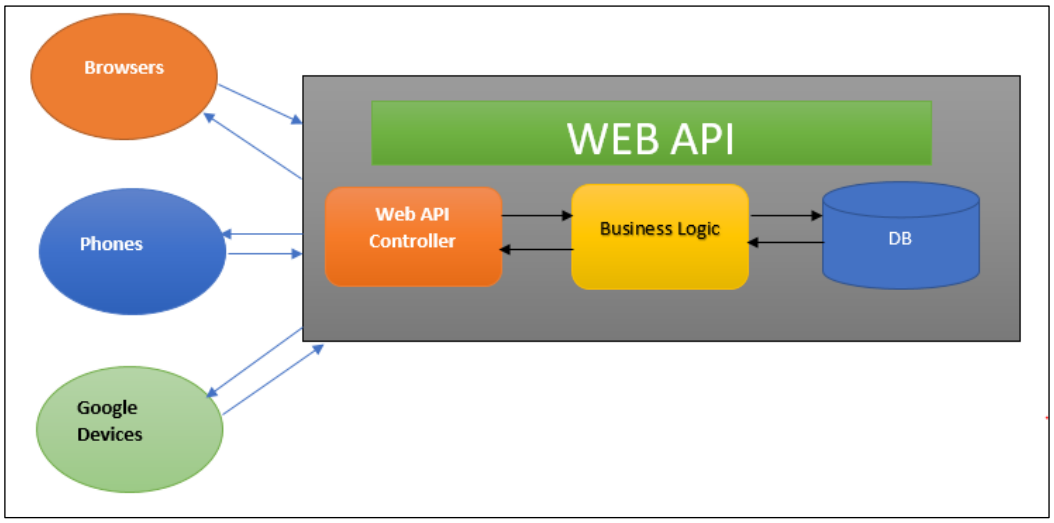


Рисунок 4.9 – Архітектура ASP.NET Core Web API

Для дослідження методів доступу до бази даних було розроблено три Web API контролера:

– EntityFrameworkController;
– DapperController;
– ADOController.

В кожному контролері є точки входу. Точка входу (або "entry point") веб-API (API, інтерфейс програмування додатків) - це кінцева точка, яка дозволяє клієнтській програмі взаємодіяти з функціональністю веб-API. Це може бути конкретна URL-адреса, за якою клієнтська програма відправляє запити на сервер з використанням певних HTTP-методів, таких як GET, POST, PUT, DELETE і т.д.

Загалом, точка входу - це місце, де клієнтська програма починає взаємодію з API і надсилає запити на сервер. Зазвичай, точка входу включає інформацію про те, які ресурси доступні через API і яким чином ці ресурси можуть бути отримані або змінені. Точка входу також може вимагати авторизації, щоб захистити конфіденційну інформацію та контролювати доступ до API.

Для візуалізації і тестуваня розроблених контролерів використовується Swagger.

Swagger (також званий OpenAPI) – це інструмент для створення, документування та використання веб-API. Swagger дозволяє описати структуру API та його операції в машиночитаній формі, що спрощує роботу з API для розробників та інтеграційних партнерів.

Основні функції Swagger включають:

- опис структури та функціональності API з використанням специфікації OpenAPI;
- генерація інтерактивної документації API, яка може бути використана для тестування та вивчення API;
- автоматична генерація клієнтських бібліотек для роботи з API різними мовами програмування;
- валідація запитів та відповідей API на відповідність стандарту OpenAPI.

Swagger також надає безліч інструментів для спрощення роботи з API, включаючи консоль для тестування запитів та відповідей, генератори коду та інтеграцію з різними інструментами розробки та тестування. Swagger є популярним інструментом для роботи з веб-API та широко використовується в індустрії.

У перечислених вище контролерах розроблені точки входу з однаковими іменами, це зроблено для того, щоб можна було порівняти результати однакової

роботи для різних методів доступу. Як виглядають контролери з точками входу у свагері можно побачити на рисунку 4.10.

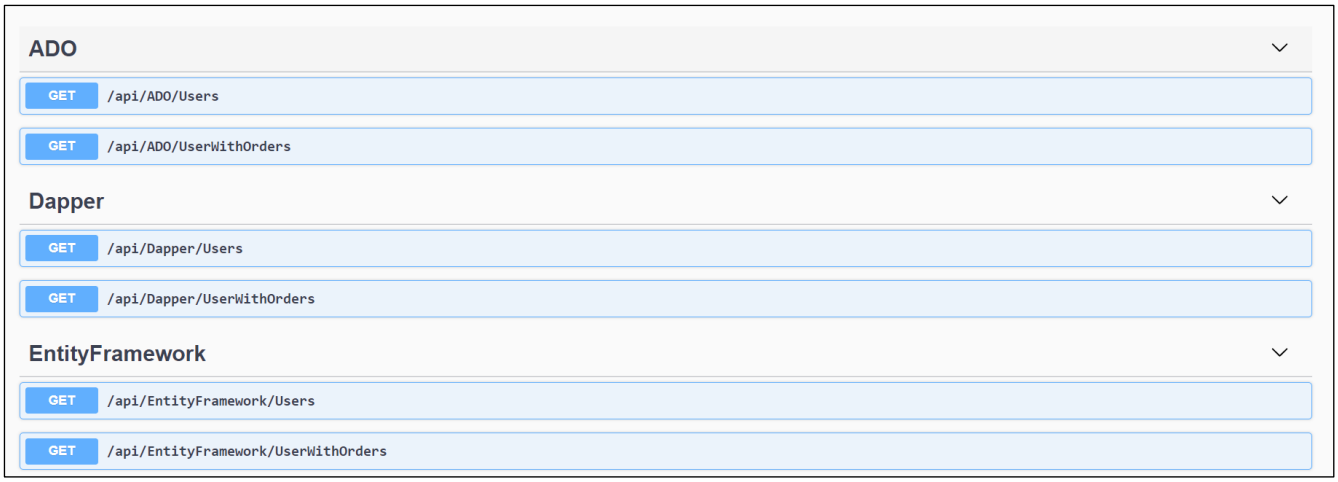


Рисунок 4.10 – Розроблені контролери з точками входу

4.3 Розробка запитів для дослідження

Для дослідження методів доступу до бази даних було розроблено запити до бази даних, які будуть використовувати різні методи доступів. Наприклад, крім круд операцій, таких як взяти усі поля користувачів, замовлень, товарів, було розроблено комплексну операцію SELECT, яка повинна буде брати дані з шести таблиць, порівнювати їх та групувати. Приклад простого запиту для отримання користувачів для Entity Framework представлений на рисунку 4.11, такий самий запит для Dapper та ADO.NET знаходиться на рисунку 4.12 та на рисунку 4.13 аналогічно.

```
1 reference | rasul.ramazanov2017@gmail.com, 43 days ago | 1 author, 2 changes
public IEnumerable<User> GetUsers()
{
    var result =_context.Users;

    return result;
}
```

Рисунок 4.11 – Запит на отримання користувачів у EF

```
1 reference | rasul.ramazanov2017@gmail.com, 47 days ago | 1 author, 1 change
public List<User> GetUsers()
{
    using (IDbConnection db = new SqlConnection(connectionString))
    {
        return db.Query<User>("SELECT * FROM Users").ToList();
    }
}
```

Рисунок 4.12 – Запит на отримання користувачів у Dapper

```
1 reference | rasul.ramazanov2017@gmail.com, 47 days ago | 1 author, 1 change
public List<User> GetUsers()
{
    var users = new List<User>();
    string sqlExpression = "SELECT * FROM Users";
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        var reader = command.ExecuteReader();

        if (reader.HasRows)
        {
            while (reader.Read()) // read data line by line
            {
                users.Add(new User {
                    Id = (int)reader.GetValue(0),
                    Name = (string)reader.GetValue(1),
                    Email = (string)reader.GetValue(2),
                    Phone = (string)reader.GetValue(3)
                });
            }
        }

        reader.Close();

        return users;
    }
}
```

Рисунок 4.13 – Запит на отримання користувачів у ADO.NET

Приклад складного запиту можна побачити на рисунку 4.14. За допомогою цього запиту можна отримати згруповану інформацію по користувачам та назву продуктів, які входять до замовлення та їхня кількість.

```
string sqlExpression = @"SELECT users.Name as UserName, products.Id as ProductId,
    products.Name as ProductName, orders.Count, seasons.Date

    FROM Users as users,
    Orders as orders,
    Products as products,
    Goods as goods,
    Catalogs as catalogs,
    Seasons as seasons

    WHERE orders.UserId = users.Id
    AND orders.GoodId = goods.Id
    AND goods.ProductId = products.Id
    AND goods.CatalogId = catalogs.Id
    AND catalogs.SeasonId = seasons.Id
    AND seasons.Date < GETDATE()

    GROUP BY users.Name, products.Id, products.Name, orders.Count, seasons.Date";
```

Рисунок 4.14 – Запит по замовленням користувачів

Якщо Dapper та ADO.NET необхідно написати запит мовою SQL, то для Entity Framework запит по замовленням користувачів буде виглядати як на рисунку 4.15.

```
1 reference | rasul.ramazanov2017@gmail.com, 43 days ago | 1 author, 2 changes
public IEnumerable<UserWithOrders> GetComplexUsers()
{
    var result = from user in _context.Set<User>()

                 join order in _context.Set<Order>() on user.Id equals order.UserId
                 join good in _context.Set<Good>() on order.GoodId equals good.Id
                 join product in _context.Set<Product>() on good.ProductId equals product.Id
                 join catalog in _context.Set<Catalog>() on good.CatalogId equals catalog.Id
                 join season in _context.Set<Season>() on catalog.SeasonId equals season.Id

                 where season.Date < DateTime.Now

                 select new UserWithOrders {
                     UserName = user.Name,
                     ProductId = product.Id,
                     ProductName = product.Name,
                     Count = order.Count
                 };

    result.GroupBy(x => new { x.UserName, x.ProductId, x.ProductName, x.Count});

    return result;
}
```

Рисунок 4.15 – Запит по замовленням користувачів у EF

Для того, щоб розуміти скільки товарів є у кожному каталозі та до якого сезону він відноситься, був розроблений запит (див. рис 4.16). На цьому запиті

можна побачити ім’я сезону, його дату початку, ідентифікатор каталогу та кількість товарів для обраного каталогу.

```
SELECT seasons.Name AS SeasonName, seasons.Date, catalogs.Id AS CatalogId, COUNT(goods.CatalogId) AS GoodsCount
            FROM Seasons AS seasons
            JOIN Catalogs AS catalogs ON catalogs.SeasonId = seasons.Id
            JOIN Goods AS goods ON goods.CatalogId = catalogs.Id
            JOIN Products AS products ON products.Id = goods.ProductId
            GROUP BY seasons.Name, seasons.Date, catalogs.Id
```

Рисунок 4.16 – Запит по товарам у каталогах

У Entity Framework попередній запит буде виглядати як показано на рисунку 4.17.

```
0 references | 0 changes | 0 authors, 0 changes
public IEnumerable<SeasonsQuerryDetails> SeasonsQuerry()
{
    var result = from product in _context.Set<Product>()

                 join good in _context.Set<Good>() on product.Id equals good.ProductId
                 join catalog in _context.Set<Catalog>() on good.CatalogId equals catalog.Id
                 join season in _context.Set<Season>() on catalog.SeasonId equals season.Id

                 where season.Date < DateTime.Now

                 select new SeasonsQuerryDetails
                 {
                     CatalogId = catalog.Id,
                     Date = season.Date,
                     SeasonName = season.Name,
                     Count = _context.Goods.Where(c => c.CatalogId == catalog.Id).Count()
                 };

    result.Load();

    result.GroupBy(x => new { x.SeasonName, x.Date, x.CatalogId });

    return result;
}
```

Рисунок 4.17 – Запит по товарам у каталогах у EF

Також буде цікавим для ведення статистики дізнатись скільки коштує уся продукція в сезоні, тому для цього був створений запит (див. рис. 4.18), який робить операцію складання Sum для необхідних в запиті продуктів.

```
SELECT seasons.Name as SeasonName, seasons.Date, catalogs.Id as CatalogId, COUNT(goods.CatalogId) as GoodsCount, SUM(products.Price)
as ProductsInSeasonTotalPrice
            FROM Seasons AS seasons
            JOIN Catalogs AS catalogs ON catalogs.SeasonId = seasons.Id
            JOIN Goods AS goods ON goods.CatalogId = catalogs.Id
            JOIN Products AS products ON products.Id = goods.ProductId

            GROUP BY seasons.Name, seasons.Date, catalogs.Id;
```

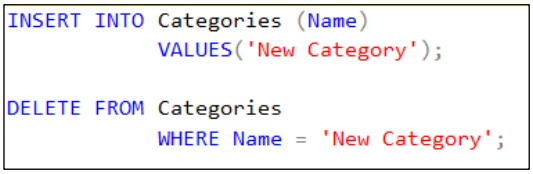
Рисунок 4.18 – Запит коштовності сезонів

У Entity Framework попередній запит буде віглядати як показано на рисунку 4.19.

```
1 reference | 0 changes | 0 authors, 0 changes
public IEnumerable<SeasonsQuerryFullDetails> GetSeasonTotalPrice()
{
    var result = from product in _context.Set<Product>()

                 join good in _context.Set<Good>() on product.Id equals good.ProductId
                 join catalog in _context.Set<Catalog>() on good.CatalogId equals catalog.Id
                 join season in _context.Set<Season>() on catalog.SeasonId equals season.Id

                 where season.Date < DateTime.Now

                 select new SeasonsQuerryFullDetails
                 {
                     CatalogId = catalog.Id,
                     Date = season.Date,
                     SeasonName = season.Name,
                     Count = _context.Goods.Where(c => c.CatalogId == catalog.Id).Count(),
                     Sum = _context.Goods.Where(c => c.ProductId == product.Id).Select(p => p.Product.Price).Sum()
                 };

    result.Load();

    result.GroupBy(x => new { x.SeasonName, x.Date, x.CatalogId });

    return result;
}
```

Рисунок 4.19 – Запит коштовності сезонів у EF

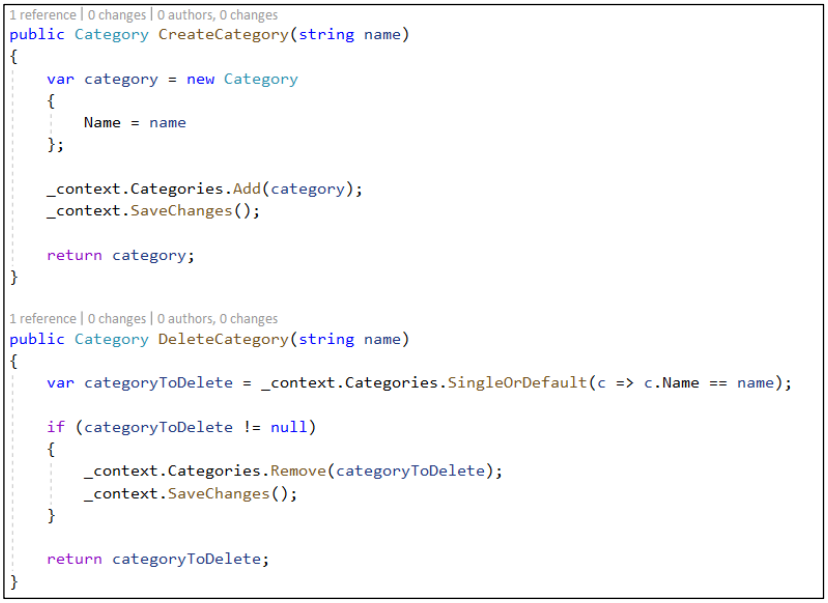
Далі буде описані запити для створення та видалення категорії (див. рис. 4.20).



```
INSERT INTO Categories (Name)
            VALUES('New Category');

DELETE FROM Categories
            WHERE Name = 'New Category';
```

Рисунок 4.20 – Запити створення та видалення категорії

Їх реалізацію у Entity Framework можна побачити на рисунку 4.21.



```
1 reference | 0 changes | 0 authors, 0 changes
public Category CreateCategory(string name)
{
    var category = new Category
    {
        Name = name
    };

    _context.Categories.Add(category);
    _context.SaveChanges();

    return category;
}

1 reference | 0 changes | 0 authors, 0 changes
public Category DeleteCategory(string name)
{
    var categoryToDelete = _context.Categories.SingleOrDefault(c => c.Name == name);

    if (categoryToDelete != null)
    {
        _context.Categories.Remove(categoryToDelete);
        _context.SaveChanges();
    }

    return categoryToDelete;
}
```

Рисунок 4.21 – Запити створення та видалення категорії у EF

5 ОПИС ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

5.1 Результати проведення експерименту

Далі будуть представлені результати виконання запитів для Entity Framework, Dapper та ADO.NET у табличному вигляді для кожного запиту, який представляє собою унікальність в досліджені.

Для проведення чистого дослідження буде відключено кешування для Entity Framework, бо завдяки йому він може виконати один і той самий запит за дуже маленький час, що є однією з переваг ORM.

Також у дослідженні для Dapper та ADO.NET були заміряні метрики з урахуванням витрачених ресурсів на підключення до бази даних та без урахування на це підключення, бо завдяки контексту ORM Entity Framework, коли ви створюєте екземпляр DbContext у своєму коді, EF створює з'єднання з базою даних і відкриває його, щоб забезпечити доступ до даних, тобто на момент виконання запитів у стистемі, Entity Framework вже має підключення до бази даних.

Далі будуть перелічені запити з їхніми назвами та вмістом, які були досліджені:

- GetUsers, запит можна побачити на рисунку 5.1;
- GetUserWithOrders, запит можна побачити на рисунку 5.2;
- GetSeasonsQuerry, запит можна побачити на рисунку 5.3;
- GetSeasonTotalPrice, запит можна побачити на рисунку 5.4;
- CreateCategory, запит можна побачити на рисунку 5.5;
- DeleteCategory, запит можна побачити на рисунку 5.6.

```
SELECT * FROM Users
```

Рисунок 5.1 – GetUsers

На запиті GetUsers можна побачити усі поля таблиці Users.

```
SELECT users.Name as UserName, products.Id as ProductId,
                products.Name as ProductName, orders.Count, seasons.Date

                FROM Users as users,
                Orders as orders,
                Products as products,
                Goods as goods,
                Catalogs as catalogs,
                Seasons as seasons

                WHERE orders.UserId = users.Id
                AND orders.GoodId = goods.Id
                AND goods.ProductId = products.Id
                AND goods.CatalogId = catalogs.Id
                AND catalogs.SeasonId = seasons.Id
                AND seasons.Date < GETDATE()

                GROUP BY users.Name, products.Id, products.Name, orders.Count, seasons.Date
```

Рисунок 5.2 – GetUserWithOrders

Запит GetUserWithOrders бере дані з таблиць Users, Orders, Products, Goods Catalogs та Seasons, та є складнішим від запиту GetUsers.

```
SELECT seasons.Name AS SeasonName, seasons.Date, catalogs.Id AS CatalogId, COUNT(goods.CatalogId) AS GoodsCount
            FROM Seasons AS seasons
            JOIN Catalogs AS catalogs ON catalogs.SeasonId = seasons.Id
            JOIN Goods AS goods ON goods.CatalogId = catalogs.Id
            JOIN Products AS products ON products.Id = goods.ProductId
            GROUP BY seasons.Name, seasons.Date, catalogs.Id
```

Рисунок 5.3 – GetSeasonsQuerry

Завдяки запиту GetSeasonsQuerry можна отримати кільскість товарів, які присутні у сезоні. У цьому запиті присутня функція агрегації Count(), яка порахує товари у сезоні.

```
SELECT seasons.Name as SeasonName, seasons.Date, catalogs.Id as CatalogId, COUNT(goods.CatalogId) as GoodsCount, SUM(products.Price)
as ProductsInSeasonTotalPrice
            FROM Seasons AS seasons
            JOIN Catalogs AS catalogs ON catalogs.SeasonId = seasons.Id
            JOIN Goods AS goods ON goods.CatalogId = catalogs.Id
            JOIN Products AS products ON products.Id = goods.ProductId

            GROUP BY seasons.Name, seasons.Date, catalogs.Id;
```

Рисунок 5.4 – GetSeasonTotalPrice

Завдяки запиту GetSeasonsQuerry можна отримати кільскість товарів, які присутні у сезоні та порахувати спільну суму цих товарів у сезоні цілком. У цьому

запиті присутні функція агрегації Count(), яка порахує товари у сезоні та функція агрегації Sum(), яка порахує спільну суму для товарів у сезоні.

```
INSERT INTO Categories (Name)
            VALUES('New Category');
```

Рисунок 5.5 – CreateCategory

Запит CreateCategory створює нову категорію з заданим іменем.

```
DELETE FROM Categories
            WHERE Name = 'New Category';
```

Рисунок 5.6 – DeleteCategory

Запит DeleteCategory видаляє категорію з заданим іменем.

Перший запит до бази даних виконується довше за інші послідуючі запити, бо перед тим, як виконати запит на сервер бази даних, серверу може знадобитися виконати ряд додаткових завдань, таких як підготовка плану виконання запиту та оптимізація запиту. Підготовка плану виконання запиту - це процес створення плану дій, який буде необхідний виконання запиту. Він включає вибір способу доступу до таблиць бази даних, вибір методу з'єднання таблиць і вибір порядку виконання операцій в запиті. Оптимізація запиту – це процес покращення продуктивності запиту шляхом його зміни або використання додаткових індексів, щоб прискорити пошук та вибірку даних.

На рисунку 5.7 можна побачити результати експерименту запиту GetUsers. На цьому рисунку видно, що середній час виконання, який замірюється у мілісекундах дорівнює одиниці у всіх технологіях доступу, враховуючи виконання запиту з підключенням до серверу бази даних і без. Це значить, що запит виконується занадто швидко, щоб можно було заміряти його у мс, що показує нам як вигідно користуватись більш чіткими одиницями заміру, як тіки.

Перший запуск запиту виконується замітно довше ніж усі інші запити. Якщо брати відношення швидкості першого запиту у тіках, то у ADO.NET w/o connection швидкість виконання є найкращою, після нього йде Entity Framework та ADO.NET,

потім по швидкості виконання йде Dapper w/o connection та Dapper. Це означає, що серверу знадобилось меньше часу для обробки першого запиту у ADO.NET w/o connection, ніж для інших технологій доступу. Також це можна спостерігати за витратою на перший запуск у мс.

Далі можна порівняти середній час виконання у тіках. На цьому етапі порівняння змінюється те, що середній час виконання для ADO.NET та ADO.NET w/o connection буде швидшим за Entity Framework. Dapper на цьому експерименті залишається найдовшим у виконанні.

Запит	GetUsers				
	Entity Framework	Dapper	Dapper W/O connection	ADO.NET	ADO.NET W/O connection
Перший запуск (тіки)	2005217	3851348	3392629	3216040	206956
Перший запуск (мс)	182	400	352	321	20
Середній час виконання (тіки)	5623	5806	6181	4702	3209
Середній час виконання (мс)	1	1	1	1	1
Затрачено пам'яті (байт)	8512	4571	4384	4544	704

Рисунок 5.7 – Результати експерименту з GetUsers

Тепер можна порівняти скільки ресурсів було затрачено для виконання запиту. ADO.NET w/o connection затратив найпомітніше менше ресурсів, а саме 704 байти для цього. Потім йде Dapper w/o connection, за ним ADO.NET і Dapper, а найбільше ресурсів потребував Entity Framework, майже вдвіче більше, ніж ADO.NET і Dapper.

На рисунку 5.8 можна побачити результати експерименту з запитом GetUsersWithOrders. На цьому рисунку також можна побачити, що середній час виконання у мс дорівнює 1 у всіх технологій доступу, що значить, що виконання цього запиту проходить швидше, ніж можна виміряти у мс. Далі можна розглянути скільки часу у тіках займає перший запуск запиту, та сама ситуація як і з запитом GetUsers, а саме найшвидшим виступає ADO.NET w/o connection, потім Entity Framework та ADO.NET, а далі Dapper w/o connection та Dapper. Те саме можна побачити і в швидкості виконання у мс. Але середній час виконання запиту відрізняється. На першому місці ADO.NET w/o connection, потім йде ADO.NET, Dapper w/o connection та Dapper, а найдовше часу потрібно було для виконання цього запиту для Entity Framework. По Витраченим ресурсам у байтах можно

виявити, що ADO.NET з підключенням та без вимагає набагато меньше пам’яті ніж Dapper, а Entity Framework витрачає найбільше цього ресурсу.

Запит	GetUserWithOrders				
	Entity Framework	Dapper	Dapper W/O connection	ADO.NET	ADO.NET W/O connection
Перший запуск (тіки)	2555442	4138181	4002818	3283229	244381
Перший запуск (мс)	260	391	352	346	27
Середній час виконання (тіки)	15413	12814	11782	11867	8633
Середній час виконання (мс)	1	1	1	1	1
Затрачено пам'яті (байт)	12496	10552	10329	1731	1324

Рисунок 5.8 – Результати експерименту GetUsersWithOrders

На рисунку 5.9 можна побачити результати експерименту запиту GetSeasonsQuerry. На цьому рисунку також можна побачити, що середній час виконання запиту проходить швидше, ніж можна виміряти у мс і у всіх технологіях стоїть 1 мс. По швидкості виконання першого запуску також аналогічна ситуація із запитами GetUsers та GetUsersWithOrders. Але по середньому часу виконання можна підкреслити, що ADO.NET w/o connection та Dapper w/o connection виконуються найшвидше, потім йдуть ADO.NET та Dapper, і з помітним відставанням у швидкості виконання запиту йде Entity Framework, який виконується у майже 9 разів довше, ніж ADO.NET w/o connection. По використаним ресурсам можна також сказати, що ADO.NET та Dapper використовують у 3 рази меньше ресурсів (байтів), ніж Entity Framework.

Запит	GetSeasonsQuerry				
	Entity Framework	Dapper	Dapper W/O connection	ADO.NET	ADO.NET W/O connection
Перший запуск (тіки)	2510343	4273887	3499499	3447487	246980
Перший запуск (мс)	248	414	355	351	40
Середній час виконання (тіки)	44839	8590	6888	7092	5319
Середній час виконання (мс)	1	1	1	1	1
Затрачено пам'яті (байт)	5048	1720	1688	1688	896

Рисунок 5.9 – Результати експерименту GetSeasonsQuerry

На рисунку 5.10 можна побачити результати експерименту запиту GetSeasonsTotalPrice. Середній час виконання запиту проходить швидше, ніж можна виміряти у мс і у всіх технологіях стоїть 1 мс. По швидкості виконання першого запуску також аналогічна ситуація як і з попередніми запитами, порівнювати можна як у тіках, так і в мс. По середньому часу виконання лідує

ADO.NET w/o connection та ADO.NET, потім йдуть Dapper w/o connection та Dapper, і найдовше виконував запит Entity Framework, який відстав від ADO.NET w/o connection майже у 2 рази.

Запит	GetSeasonTotalPrice				
	Entity Framework	Dapper	Dapper W/O connection	ADO.NET	ADO.NET W/O connection
Перший запуск (тіки)	2983407	4065444	3811651	4184960	199574
Перший запуск (мс)	281	441	359	349	27
Середній час виконання (тіки)	13621	9117	9328	8586	7400
Середній час виконання (мс)	1	1	1	1	1
Затрачено пам'яті (байт)	9992	3022	2978	2764	1586

Рисунок 5.10 – Результати експерименту GetSeasonsTotalPrice

На рисунку 5.11 можна побачити результати експерименту запиту CreateCategory. Час виконання першого запуску, як і в попередніх запитах, виконується найдовше та розподілити по швидкості можна так: ADO.NET w/o connection, Entity Framework, ADO.NET, Dapper w/o connection та Dapper. По середньому часу виконання у мс вже можна побачити, що Entity Framework помітно відстає. Більш точну пропорцію можна зробити порівнявши середній час виконання у тіках, де видно, що ADO.NET w/o connection швидший у 2.6 разів, ніж Entity Framework. По використаній пам’яті аналогічно попереднім запитам, де Entity Framework використовує у 3 рази більше ресурсів, ніж інші технології доступу.

Запит	CreateCategory				
	Entity Framework	Dapper	Dapper W/O connection	ADO.NET	ADO.NET W/O connection
Перший запуск (тіки)	1592580	3900297	3601395	3294704	214774
Перший запуск (мс)	161	421	329	313	16
Середній час виконання (тіки)	26597	11690	11098	10710	9817
Середній час виконання (мс)	2.5	1.5	1	1	1
Затрачено пам'яті (байт)	7664	2554	2474	2468	1238

Рисунок 5.11 – Результати експерименту CreateCategory

На рисунку 5.12 можна побачити результати експерименту запиту DeleteCategory. Час виконання першого запуску такий самий як і в запитах раніше, якщо дивитись на те, яка технологія виконала його найшвидше. Але по середньому часу виконання тепер Entity Framework найшвидший, де він у 5 разів швидший аніж Dapper , та у 4 рази швидше, ніж ADO.NET, що можна побачити як у середньому

часі виконання у тіках, так і в мс. По використаній пам’яті все також аналогічно попереднім запитам, де Entity Framework використовує в рази більше пам’яті, ніж інші досліджувані технології.

Запит	DeleteCategory				
	Entity Framework	Dapper	Dapper W/O connection	ADO.NET	ADO.NET W/O connection
Перший запуск (тіки)	2949541	3714834	3741236	3655405	196057
Перший запуск (мс)	273	375	318	329	17
Середній час виконання (тіки)	15092	77176	73560	71484	63049
Середній час виконання (мс)	1.3	7	6.9	6.4	6.7
Затрачено пам'яті (байт)	9064	2288	1934	936	772

Рисунок 5.12 – Результати експерименту DeleteCategory

Також слід зауважити, що для проведення експерименту у Entity Framework була відключена функція «Lazy load» та відключена змога кешувати запити, тобто кожен запит він виконував як вперше. Якби функція кешування була увімкнена, то при підключені до серверу, при повторному запиту він би витрачав мінімальну кількість часу. Тому це треба мати на увазі, роблячи висновки про ці технології.

5.2 Висновки та рекомендації з експериментального дослідження

Після виконання експерементів було з’ясовано як складність виконання запиту впливає на час його виконання. Як впливають на час винанання також типи запитів. Були розглянуті реалізації запитів для технологій доступу до бази даних та було проведено опис запитів для експерементів.

Кожну технологію доступу можна описати окремо, бо вони дуже відрізняються по наданному функціоналу та використанню.

ADO.NET показав, що він займає дуже мало ресурсів, порівняно з ORM та Micro ORM, виконання запиту проходить чи не найшвидше майже у всіх випадках. Проблеми можуть виникати під час написання запиту на мові SQL, тобто для використання цієї технології обов’язково треба мати досвід роботи з SQL. Інша проблема може виникати під час того, як треба взяти дані, які повернулись з бази даних, бо внутрішнього автомапінгу у ADO.NET немає, тому тут треба також бути пильним.

Dapper дуже гарно себе показав у швидкості виконання запитів, де він майже не відставав від ADO.NET і переважав у декілька разів Entity Framework. По витраченим ресурсам можна побачити чому Dapper вважеється легковісною Micro ORM, витрачає він ресурсів майже так саме як і ADO.NET. Як і в ADO.NET, для використання Dapper необхідно знати SQL, але він не має проблем з автомапінгом вилучених даних, що є дуже корисним інструментом, який дозволяє навіть заповнювати поля в користувацьких класах з вилучених Dapper даних, які належать іншим классам.

Entity Framework є дуже популярною ORM і це цілком підтверджено експерементами чому саме так. Завдяки внутрішному функціоналу можна використовувати різні підходи розробки, такі як Code First, DataBase та First Model First, робити міграції бази даних, легко писати запити, які ORM самотушки відправить до серверу бази даних. Але є і мінуси у цьому, як можна було побачити, чим складніший запит, тим більше часу необхідно було на його використання Entity Framework у порівнянні співвідношень до часу виконання з іншими технологіями доступу. Коли запити прості, Entity Framework витрачав на це часу меньше ніж Dapper, що є дуже гарним показником.

Таким чином, якщо у вас є простий додаток або ви хочете максимально прискорити виконання запитів до бази даних, можете використовувати Dapper. Якщо у вас велика і складна програма і ви хочете використовувати об'єктно-орієнтований підхід, то Entity Framework може бути найкращим вибором. ADO.NET може використовуватися для виконання більш складних запитів та керування транзакціями [32].

ВИСНОВКИ

В результаті виконання даної кваліфікаційної роботи було досягнуто поставлену мету, а саме проведено аналіз технологій доступу до реляційних баз даних для обраної СУБД MS SQL та обрано шляхом вирішення багатокритеріальної задачі найбільш підходящі для дослідження такі технології доступу, як ORM Entity Framework, Micro ORM Dapper, ADO.NET.

Проведено планування експериментального дослідження, а саме обрано предметну область та розроблено схему БД, обрано метрики для оцінки результатів експериментів, такі як швидкість виконання запитів у тіках та мс, кількість витраченної оперативної пам’яті у байтах та кількість коду, яку необхідно написати. Реалізовано програмне рішення під обрані технології доступу до бази даних.

Були виконані експеременти на запланованих для дослідження запитах, були проаналізовані результати дослідження та написані рекомендації щодо використання досліджуваних технологій доступу до реляційних баз даних.

При виконанні даної кваліфікаційної роботи були успішно реалізовані всі вимоги, сформульовані в постановці завдання.

За результатами дослідження було опубліковано тези «Дослідження методів доступу до реляційних баз даних» на двадцять сьомий міжнародний молодіжний форум «РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ В XXI ст.», а також, підготовлено до подачі наукову статтю «Дослідження технологій доступу до реляційних баз даних» у науковому журналі «Біоніка інтелекту» (див. додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бьюли А. Изучаем SQL / Алан Бьюли. - Символ-Плюс, 2007. - 112 с.

2. Молинаро Энтони. SQL. Сборник рецептов / Энтони Молинаро. - O'Reilly, 2009. - 372 с.

3. Бураков П.В. Введение в системы баз данных. Учебное пособие / П.В. Бураков., В.Ю. Петров. - СПбГУ ИТМО, 2010. 74 с.

4. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#. 4-е изд. / Джеффри Рихтер. - Питер, 2013. - 532 с.

5. Троелсен, Е. C # і платформа .NET. Бібліотека програміста [Текст] / Е. Троелсен - СПб .: Питер, 2013. - 755с.

6. Microsoft Developer Network. Библиотека MSDN. Разработка на .NET [Электронный ресурс] / MSDN – сеть разработчиков Microsoft. – Режим доступа : www/URL:https://msdn.microsoft.com/ru-ru/ – 22.05.2017 г. – Загл. сэкрана.

7. SQL [Электронный ресурс]: пер. с англ / – М.: ДМК Пресс, 2008. – 454 с.

8. Entity Framework Tutorial [Електроний ресурс] – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/what-is-entityframework.aspx (дата звернення: 05.12.2022 р.).

9. Performance: Entity Framework 7 vs. Dapper.net vs. raw ADO.NET [Електроний ресурс] – Режим доступу до ресурсу: http://ppanyukov.github.io/2015/05/20/entity-framework-7-performance.html (дата звернення: 12.12.2022 р.).

10. Dapper, Entity Framework and hybrid apps [Електроний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/archive/msdn-magazine/2016/may/data-points-dapper-entity-framework-and-hybrid-apps (дата звернення: 07.12.2022 р.).

11. RDBMS (relational database management system). URL: https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system (дата звернення: 12.02.2023).

12. SQL Server 2019. URL: https://www.microsoft.com/ru-ru/sql-server/sql-server-2019 (дата звернення: 07.03.2023).

13. What is SQL. URL: https://aws.amazon.com/ru/what-is/sql/ (дата звернення: 12.02.2023).

14. What Is a Database. URL: https://www.oracle.com/database/what-is-database/ (дата звернення: 07.03.2023).

15. What is a Relational Database (RDBMS). URL: https://www.oracle.com/database/what-is-a-relational-database/ (дата звернення: 10.03.2023).

16. Non-relational data and NoSQL. URL: https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data (дата звернення: 15.03.2023).

17. ADO.NET Overview. URL: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview (дата звернення: 17.03.2023).

18. What is an ORM. URL: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/ (дата звернення: 17.03.2023).

19. Introducing Mighty: A new micro-ORM for .NET Core. URL: https://medium.com/swlh/introducing-mighty-a-new-net-9bcc7bdd4814#:~:text=A%20micro%2DORM%20is%20a,Hydrating%20objects%20from%20the%20database (дата звернення: 20.03.2023).

20. Get started with Entity Framework Core. URL: https://learn.microsoft.com/ru-ru/ef/core/get-started/overview/first-app?tabs=netcore-cli (дата звернення: 20.03.2023).

21. NHibernate – Overview. URL: https://www.tutorialspoint.com/nhibernate/nhibernate_overview.htm (дата звернення: 20.03.2023).

22. Dapper – Micro ORM With Epic Possibilities. URL: https://methodpoet.com/dapper/ (дата звернення: 21.03.2023).

23. ORM Library for .NET & .NET Core - XPO – DevExpress. URL: https://www.devexpress.com/products/net/orm/#:~:text=XPO%20is%20an%20Object%2DRelational,By%20using%20. (дата звернення: 23.03.2023).

24. DataObjects.Net - Overview. URL: https://dataobjects.net/ (дата звернення: 23.03.2023).

25. DateTime.Ticks. URL: https://learn.microsoft.com/en-us/dotnet/api/system.datetime.ticks?view=net-8.0 (дата звернення: 23.03.2023).

26. Ecommerce Defined. URL: https://www.investopedia.com/terms/e/ecommerce.asp (дата звернення: 23.03.2023).

27. Code First to a New Database. URL: https://learn.microsoft.com/ru-ru/ef/ef6/modeling/code-first/workflows/new-database (дата звернення: 23.03.2023).

28. Database First. URL: https://learn.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first (дата звернення: 23.03.2023).

29. Model First. URL: https://learn.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/model-first (дата звернення: 23.03.2023).

30. DBContext creation. URL: https://learn.microsoft.com/ru-ru/ef/core/cli/dbcontext-creation?source=recommendations&tabs=dotnet-core-cli (дата звернення: 23.03.2023).

31. Mazurova, O. Research of ACID transaction implementation methods for distributed databases using replication technology / Mazurova, O., Naboka, A., Shirokopetleva, M. Innovative technologies and scientific solutions for industries, (2 (16), pp. 19-31. Doi: 10.30837/ITSSI.2021.16.019.

32. Рамазанов Р.Ш. Дослідження технологій доступу до реляційних баз даних // 27-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6. – Харків: ХНУРЕ. 2023., с. 341-342.