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 методів зберігання великого обсягу
візуальних даних в базах даних
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-4

Віталій НІКОЛЕНКО
(Власне ім'я, ПРІЗВИЩЕ)

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

Тип програми освітньо-наукова

Керівник доц. Ірина КИРИЧЕНКО
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ніколенку Віталію Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів зберігання великого обсягу візуальних даних в базах даних»
Затверджена наказом по університету від 15.04.2025р. №290Ст

2. Термін подання студентом роботи до екзаменаційної комісії 16.06.2025

3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, методи зберігання даних в базах даних, бази даних MS SQL Server, MongoDB*

4. Перелік питань, що потрібно опрацювати в роботі
аналіз проблемної галузі, постановка задачі, аналіз та вибір реляційної та нереляційної баз даних, моделювання предметної області, планування експериментального дослідження, проведення дослідження, аналіз отриманих результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	16.04.2025	*виконано*
2	Аналіз предметної галузі і постановка задачі	25.04.2025	*виконано*
3	Опис прийнятих проектних рішень	02.05.2025	*виконано*
4	Проектування програмного застосунку та реалізація	16.05.2025	*виконано*
5	Опис експериментальних досліджень	23.05.2025	*виконано*
6	Підготовка до апробації результатів дослідження. Публікація матеріалів	29.05.2025	*виконано*
7	Підготовка пояснювальної записки	10.06.2025	*виконано*
8	Підготовка презентації та доповіді	10.06.2025	*виконано*
9	Перевірка на плагіат	12.06.2025	*виконано*
10	Нормоконтроль	13.06.2025	*виконано*
11	Рецензування	13.06.2025	*виконано*
12	Попередній захист	14.06.2025	*виконано*
13	Занесення диплома в електронний архів	16.06.2025	*виконано*
14	Допуск до захисту у зав. кафедри	16.06.2025	*виконано*

Дата видачі завдання 16 квітня 2025

Студент _____ Віталій НІКОЛЕНКО
(підпис)

Керівник роботи _____ доц. Ірина КИРИЧЕНКО
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 75 с., 15 рис., 10 табл., 36 джерел, 4 додатки.

АНАЛІЗ, БАЗА ДАНИХ, ВІЗУАЛЬНІ ДАНІ, ГІБРИДНЕ СХОВИЩЕ, ЕФЕКТИВНІСТЬ, ЗОБРАЖЕННЯ, НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ, ПРОДУКТИВНІСТЬ, РЕЛЯЦІЙНІ БАЗИ ДАНИХ, FILESTREAM, MONGODB, MS SQL SERVER

Об’єктом дослідження є бази даних та методи зберігання зображень в них.

Метою роботи є проведення дослідження продуктивності зберігання зображень в реляційних та нереляційних базах даних.

Методами дослідження є аналіз існуючих реляційних та нереляційних СУБД, переваг та недоліків зберігання в них візуальних даних і існуючих механізмів, інструментів та методів підвищення ефективності зберігання та обробки цих даних.

В результаті роботи було проведено еперименталье дослідження для визначення оптимальних методів зберігання великих обсягів візуальних даних в реляційних та нереляційних базах даних.

ANALYSIS, DATABASE, VISUAL DATA, HYBRID STORAGE, EFFICIENCY, IMAGE, NON-RELATIONAL DATABASES, PERFORMANCE, RELATIONAL DATABASES, FILESTREAM, MONGODB, MS SQL SERVER

The object of research is databases and methods of storing big visual data.

The purpose of the work is to conduct research on the performance of different methods of image storage in different relation and non-relational databases.

Research methods include analysis of existing relational and non-relational DBMSs, advantages and disadvantages of storing visual data in them, and existing mechanisms, tools, and methods for improving the efficiency of storing and processing this data.

As a result of the complex course project, the experimental research was conducted to determine optimal methods for storing large volumes of visual data in relational and non-relational databases.

Завідувачу кафедри

ПІ_____
(скорочена назва кафедри)

проф. Кирилу СМЕЛЯКОВУ
(вчене звання, сласне ім’я, прізвище)

ЗАЯВА

щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Ніколенко Віталій Володимирович, здобувач вищої освіти на другому (магістерському) рівні вищої освіти академічної групи ІПЗм-23-4 кафедра програмної інженерії , заявляю: моя кваліфікаційна робота на тему

Дослідження методів зберігання великого обсягу візуальних даних в базах даних,
(назва роботи)

що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії "ElArKhNURE". Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ "ElArKhNURE". Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

12.06.2025 Віталій НІКОЛЕНКО

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі і постановка задачі 10
1.1 Аналіз методів зберігання зображень 10
1.2 Аналіз методів зберігання зображень в реляційних базах даних 11
1.3 Аналіз методів зберігання зображень в нереляційних базах даних 13
1.4 Постановка задачі 15
2 Опис прийнятих проектних рішень 16
2.1 Методи вибору СУБД для дослідження 16
2.2 Аналіз та вибір реляційної СУБД для дослідження 17
2.3 Аналіз та вибір нереляційної СУБД 21
2.3 Аналіз та вибір предметної області дослідження 25
2.4 Планування експериментального дослідження 29
3 Проектування програмного застосунку та реалізація 32
3.1 Проектування програмного застосунку для виконання експериментального дослідження 32
3.2 Вибір технологій для програмної реалізації 33
3.3 Розробка серверної частини програмної системи 33
4 Опис експериментальних досліджень 43
4.1 Проведення експериментальних досліджень 43
4.2 Оцінка масштабованості методів зберігання даних 47
Висновки 52
Перелік джерел посилання 54
Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 57

ВСТУП

В теперішній час спостерігається зріст обсягу візуальних даних через поширення комп’ютерного зору, медичної діагностики, автономних систем, соціальних мереж тощо. Управління цими даними є критично важливим для забезпечення їхньої доступності, безпеки, масштабованості та ефективного використання в подальшому.

Загальноприйнятий метод зберігання візуальних даних – на окремому хмарному сховищі, а в базі даних міститься лише посилання на файл на цьому сховищі. Але існують окремі випадки, коли використання хмарних засобів неможливо:

- вимоги безпеки програмної системи – міститься конфіденційні дані чи інформація, яку необхідно зберігати лише на власній інфраструктурі програмної системи;
- вимоги ACID (atomicity, consistency, isolation, durability) – необхідність атомарності та транзакційної цілісності при роботі з даними;
- робота в ізольованих середовищах без доступу до хмарних засобів.

Актуальність роботи полягає в необхідності знаходження оптимальних підходів до зберігання візуальних даних в базах даних при різних умовах використання.

Отже, об’єктом дослідження є методи зберігання даних в реляційних та нереляційних базах даних. Предметом дослідження є продуктивність та масштабованість методів зберігання великих обсягів неструктурованих даних, таких як зображення.

Методи дослідження включають аналіз існуючих підходів до зберігання даних, моделювання предметної області та власне експериментальне дослідження цих підходів.

Метою дослідження є знаходження оптимальноих методу зберігання великого обсягу візуальних даних в реляційних та нереляційних базах даних з боку продуктивності та масштабованості в залежності від розмірі та обсягів даних.

Для досягнення поставленої мети було поставлено наступні задачі:

- проаналізувати існуючі підходи до збереження візуальних даних в реляційних та нереляційних базах даних;
- провести аналіз та вибір однієї реляційної та однієї нереляційної СУБД для подальшого дослідження;
- зробити аналіз, вибір та моделювання актуальної предметної області для дослідження;
- провести експериментальне дослідження та проаналізувати отримані результати.

Таким чином, дана робота спрямована на знаходження оптимальних методів зберігання великих обсягів даних для реляційних та нереляційних СКБД (систем керування базами даних).

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз методів зберігання зображень

З кожним днем зростає обсяг візуальної інформації, яку треба зберігати та обробляти в сучасних програмних системах. Наприклад, за 2022 в соціальній мережі Snapchat щохвилини надсилалось 2.43 мільйони «снепів» – повідомлень, що містять фото або відео [1]. За 2024 рік це число збільшилось до 3.3 мільйонів [2].

Робота з такими обсягами даних потребує оптимальних підходів до зберігання та обробки зображень. Відомо, що використання окремих сховищ є оптимальним варіантом для зберігання даних великого розміру [3].

Файлові сховища значно дешевші за бази даних. Наприклад, 1ТБ хмарного сховища на Microsoft Azure коштуватиме до $150 за преміум опцію (без урахування витрат на запис та читання даних) [4], коли як пакет послуг MongoDB зі сховищем 1ТБ коштуватиме $11 за годину, тобто $7920 за місяць [5]. Бази даних потребують великої обчислювальної потужності, тому в пакеті MongoDB відразу є й виділені віртуальні процесори та оперативна пам’ять.

Також зображення мають різний розмір файлу. Стиснуте Full HD зображення (1920x1080) може займати до 800КБ, коли як стиснуте 8K зображення (7680x4320) може бути розміром до 20МБ. Великі файли створюють фрагментацію і дискового простору хмарних сховищ, і баз даних. В хмарних сховищах фрагментація дискового простору не є великою проблемою через швидкодію доступу до файлу за прямим посиланням. А, наприклад, в реляційних базах даних таблиці даних розділено на сторінки невеликого розміру (8КБ для MS SQL Server за замовченням). Постійні запис/видалення даних можуть призвести до фрагментації, деякі сторінки будуть майже порожніми, що призведе до зниження продуктивності бази даних.

Зазвичай, програмні системи складаються з клієнтської частини, серверної частини, яка містить бізнес-логіку застосунку, серверу бази даних, на якій власне розміщено базу даних, та хмарного сховища. В базі даних містяться метадані зображення та посилання на це зображення на хмарі. Поєднання традиційних баз даних та додаткових файлових сховищ називають гібридним сховищем [6].

На рисунку 1.1 наведено загальний приклад діаграми розгортання такої програмної системи.

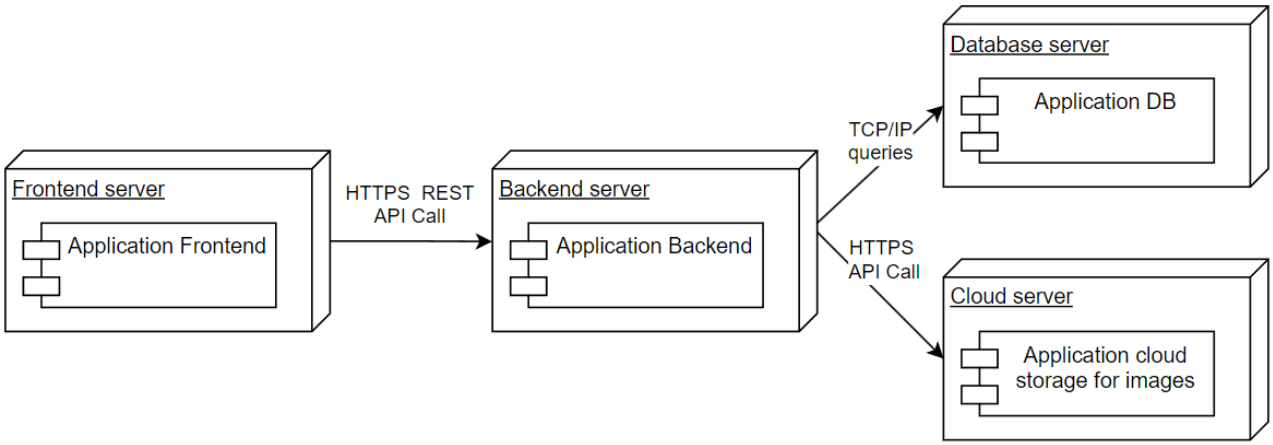

Рисунок 1.1 – Діаграма розгортання програмної системи (рисунок виконаний самостійно)

Зростання обсягів візуальної інформації вимагає вдосконалення підходів до її зберігання та обробки, тому доцільно дослідити різні методи зберігання для реляційних та нереляційних баз даних.

1.2 Аналіз методів зберігання зображень в реляційних базах даних

Реляційні бази даних, як було вказано раніше, не пристосовані до зберігання великих обсягів візуальних даних в них. Вони були спроектовані для роботи зі структурованими даними невеликого розміру. Також реляційні бази даних розраховані на вертикальне масштабування, тобто збільшення потужностей та обсягів сховища на одній машині [7]. Однак, неможливо нескінченно збільшувати потужність одного сервера. Саме тому для зберігання великих обсягів візуальної інформації необхідне горизонтальне масштабування, яке передбачає додавання нових машин в існуючу систему. В більшості з випадків нативна підтримка горизонтального масштабування для реляційних баз даних відсутня або дуже обмежена, тому необхідно використання зовнішніх інструментів, таких як Vitess.

Vitess це розподілена система баз даних, побудована на основі MySQL, яка використовується для розгортання, масштабування та управління великими кластерами баз даних. Підтримуються такі механізми масштабування, як шардінг та реплікація. Тобто цей інструмент поєднує переваги реляційних та нереляційних систем баз даних [8].

Також не всі реляційні бази даних можуть зберігати великі дані. В базах даних, що не підтримують (binary) large object (B)LOB, зображення необхідно зберігати в форматі base64, що ще додатково збільшує розмір даних.

Серед реляційних СУБД з підтримкою зберігання великих файлів можна відмітити MS SQL Server та Oracle. В MS SQL Server присутній механізм FILESTREAM, який дозволяє базі даних зберігати неструктуровані дані, такі як документи або зображення у файловій системі [9]. Для того, щоб зберігати великі файли в файловій системі бази даних, необхідно створити колонку бінарних даних varbinary(max) з атрибутом FILESTREAM.

Нижче наведено приклад коду створення таблиці.

```
CREATE TABLE Images
(
    [Id] [uniqueidentifier] ROWGUIDCOL NOT NULL UNIQUE,
    [FileName] VARCHAR(50) NOT NULL,
    [Content] VARBINARY(MAX) FILESTREAM NOT NULL
);
```

FILESTREAM рекомендується використовувати в таких випадках:

- об’єкти, що зберігаються, в більшості розміром більше ніж 1МБ;
- важлива швидкість читання файлів;
- логіку застосунку реалізовано на серверній частині застосунку, а не в базі даних.

FILESTREAM використовує системний кеш NT для кешування файлів. Кешування файлів у системі дозволяє знизити вплив зберігання файлів на продуктивність движка бази даних. Використання окремого кешу дозволяє не використовувати буферний пул SQL Server, що значить, що на запити доступно більше пам’яті бази даних.

В СУБД Oracle реалізовано аналогічний функціонал «Oracle SecureFiles» [10].

Отже, зберігання великого обсягу візуальної інформації в реляційних базах даних не є оптимальним підходом, але розробники СУБД додають різні інструменти для підвищення продуктивності.

1.3 Аналіз методів зберігання зображень в нереляційних базах даних

NoSQL (нереляційні) бази даних відрізняються від реляційних підходами до зберігання, організації, обробки та масштабування даних.

На відміну від табличок реляційних баз даних, в нереляційних базах даних дані можуть зберігатися у вигляді документів, пар ключ-значення, графів, стовпців. Гнучкість NoSQL баз даних дозволяє зберігати дані будь-якого розміру в необхідному форматі, і кожен з методів зберігання даних має свої переваги та застосування.

Нижче наведено приклад структури документу публікації, в яку вкладено зображення. Така структура дозволяє підвищити продуктивність читання даних та простоту розуміння структури даних.

```
{
  "_id": ObjectId,
  "user_id": ObjectId,
  "content": String,
  "created_at": ISODate,
  "images": [{
    "content": BLOB,
    "metadata": {
      "filename": String,
      "size": Number,
      "format": String,
      "dimensions": {
        "width": Number,
        "height": Number
      },
      "created_at": ISODate,
      "gps_location": {
        "latitude": Number,
        "longitude": Number
      },
      "tags": [String]
    }
  }
}
```

Іншою перевагою NoSQL баз даних є підтримка механізмів шардінгу та реплікації. Завдяки шардінгу, дані розподіляються між декількома серверами для горизонтального масштабування, а розподіл навантаження між кількома серверами може підвищити продуктивність всієї програмної системи. В більшості нереляційних СУБД підтримується автоматичний шардінг та балансування розподілу даних.

Реплікація – процес створення копій даних на серверах бази даних. Таким чином досягається висока доступність та відмовостійкість. Якщо один з серверів виходить з ладу, дані доступні на репліках цього сервера. Використання реплікації є обов'язковим для критичних систем, де потрібна висока доступність та цілісність даних.

В MongoDB, наприклад, можна поєднати шардінг і реплікацію, таким чином досягається висока доступність даних та продуктивність програмної системи (див. рис. 1.2). Дані розподіляються між шардами, потім проходить синхронізація цих даних між репліками [11] .

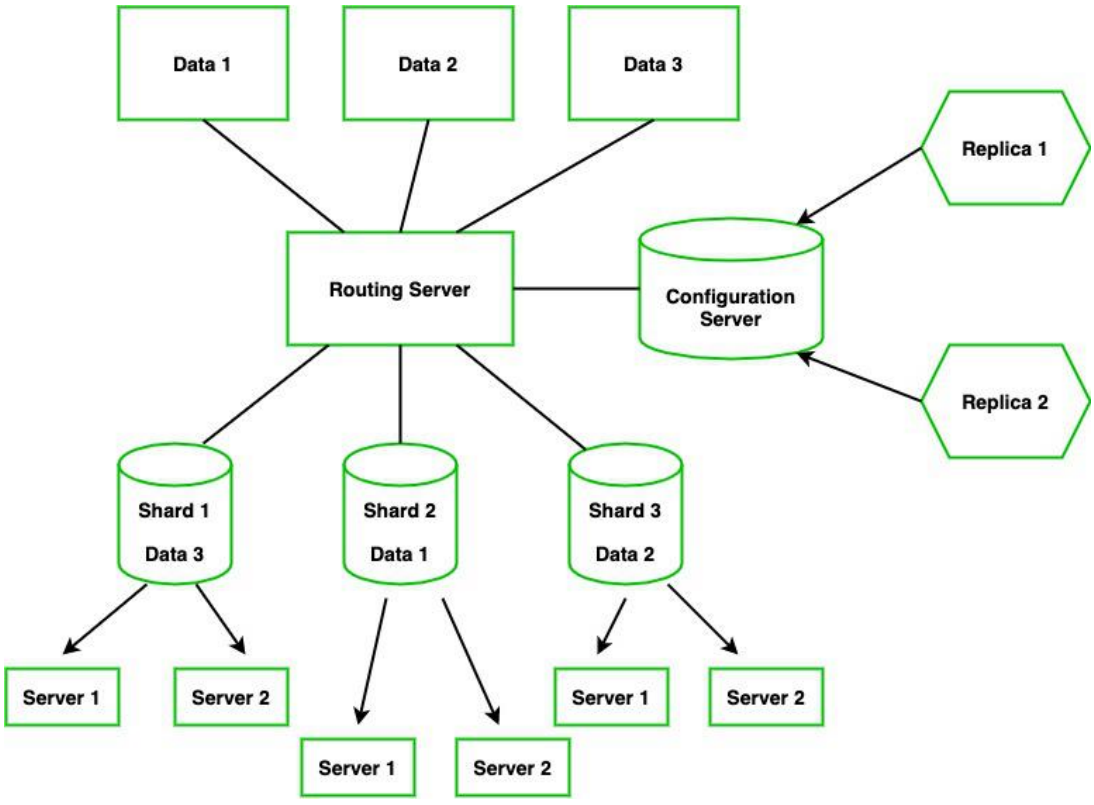

Рисунок 1.2 – Поєднання реплікації та шардінгу [11]

В цілому NoSQL бази даних спроектовані та оптимізовані для роботи з великими обсягами даних, деякі з них мають нативну підтримку різних методів інтелектуального аналізу даних.

Зважаючи на всі переваги нереляційних баз даних над реляційними, вони повинні краще підходити для зберігання великих обсягів візуальних даних.

1.4 Постановка задачі

Метою роботи є дослідження методів зберігання великого обсягу графічної інформації (зображень) в базах даних та знаходження оптимального підходу для зберігання.

Для досягнення мети дослідження необхідно виконати наступні задачі:

- провести аналіз існуючих реляційних баз даних та обрати одну з них для порівняння;
- провести аналіз та обрати нереляційну базу даних для дослідження;
- визначити предметну область для дослідження;
- спланувати експерименти з різними розмірами та кількістю зображень, щоб визначити найкращий метод зберігання для заданих параметрів;
- здійснити проєктування та реалізацію програмного забезпечення для виконання запланованих експериментів;
- виконати експериментальне дослідження;
- порівняти результати експериментів та сформулювати висновки щодо доцільності використання різних методів зберігання великого обсягу зображень у базах даних.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

2.1 Методи вибору СУБД для дослідження

Для дослідження методів зберігання великого обсягу візуальних даних вирішено обрати одну реляційну та одну нереляційну СУБД. Для розрахунку оптимального варіанту було обрано лінійну згортку з ваговими коефіцієнтами [12] :

$$Z = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij}, \tag{2.1}$$

де α_j – нормуючий коефіцієнт,

β_j – вагові коефіцієнти,

a_{ij} – значення альтернативи i за критерієм j.

Для оцінювання баз даних обрано наступні критерії:

- кількість виділених процесорів (Cores);
- обсяг виділеної оперативної пам’яті (Memory);
- максимальний розмір сховища (Storage);
- максимальний розмір файлу (File size);
- вартість за один місяць (Cost);
- можливість самостійного хостингу (Selfhosting).

Більшість нереляційних баз даних підтримують шардінг, а для реляційних СУБД було створено сторонні інструменти для забезпечення горизонтального масштабування, тому його не було обрано як критерій.

Методи забезпечення безпеки даних (шифрування, резервне копіювання тощо) реалізовані в більшості баз даних. Деякі методи шифрування, частота створення резервних копій тощо залежать від постачальника сервісу та обраних параметрів СУБД, тому цей критерій також опущено.

При самостійному хостингу такі значення, як кількість процесорів, пам’ять тощо обмежені лише власним апаратним забезпеченням, тому їх буде отримано з пакетів послуг популярних хостингів.

Також необхідно визначити вагові коефіцієнти для обраних критеріїв:

- кількість виділених процесорів (Cores) – 0.1. Цей параметр впливає на швидкодію бази даних та кількість одночасно виконуваних запитів. В залежності від предметної області, вимоги до продуктивності можуть сильно відрізнятися. Також цей показник є менш об’єктивним, тому що це залежить від обраного хостингу/власного апаратного забезпечення;
- обсяг виділеної оперативної пам’яті (Memory) – 0.1, аналогічно до критерія вище;
- максимальний розмір одного сховища (Storage) – 0.2. Параметр вертикального масштабування. Чим більше необхідно окремих машин для горизонтального масштабування, тим складніше підтримувати цілісність системи;
- максимальний розмір файлу (File size) – 0.2. Важливо, щоб в СУБД була можливість зберігати зображення будь-якого розміру;
- вартість за один місяць (Cost) – 0.15. Зберігання великої кількості даних в базах даних дорожче ніж використання хмарних сховищ, саме тому вартість є важливою складовою вибору СУБД;
- можливість самостійного хостингу – 0.25. Важливий параметр саме для цього дослідження у випадку, коли необхідно розміщувати базу даних на власних машинах.

За результатами згортки буде обрано оптимальні СУБД для подальшого експериментального дослідження.

2.2 Аналіз та вибір реляційної СУБД для дослідження

Серед реляційних баз даних розглянуто наступні:

- MySQL – одна з найпопулярніших реляційних СУБД, має підтримку типів даних BLOB та LARGEBLOB для великих файлів. Завдяки своїй

популярності, має багато сторонніх інструментів для горизонтального масштабування [13];

- MariaDB – відгалуження MySQL, створене в 2009 році після придбання останньої компанією Oracle. Додатково оптимізована для роботи з великими даними, завдяки механізму зберігання даних XtraDB, який підтримує типи даних BLOB та має механізм стиснення [14];
- MS SQL Server – високопродуктивна реляційна СУБД від Microsoft, має тип даних varbinary для зберігання даних великого розміру. Також присутній механізм FILESTREAM, що дозволяє зберігати файли не в базі даних, а в файловому просторі сервера бази даних. Є можливість інтеграції з іншими хмарними сервісами Azure [15];
- Oracle Database – популярна СУБД для комерційного використання, нативно підтримує шардінг та реплікацію. Реалізовано функціонал SecureFiles, аналогічний FILESTREAM від MS SQL Server. Також доступна інтеграція з іншими хмарними сервісами Oracle [16];
- IBM Db2 – реляційна СУБД, яка окрім перевищених вище функцій інших СУБД має механізми стиснення BLOB та дедуплікації даних. Також є можливість розміщувати модулі машинного навчання прямо в базі даних [17].

Обрано наступні пакети хостингів зі сховищем від 1ТБ та 16 ядрами процесора:

- MySQL – Google Cloud MySQL [18];
- MariaDB – Azure Database for MariaDB [19];
- MS SQL Server – Azure Virtual Machines for SQL Server [20];
- Oracle Database – Oracle Autonomous Database [21];
- IBM Db2 - Amazon RDS for Db2 [22].

Отримано дані за критеріями, переліченими в пункті 2.1.

Кількість виділених процесорів – 16 для всіх альтернатив.

Обсяг виділеної оперативної пам’яті, ГБ:

- MySQL – 96;

– MariaDB – 80;

– MS SQL Server – 64;

– Oracle Database – 32;

– IBM Db2 – 64.

Розмір сховища – 1 ТБ для всіх альтернатив. Кожна з альтернатив також підтримує вертикальне масштабування за необхідністю.

Максимальний розмір файлу, МБ:

– MySQL – 4096;

– MariaDB – 4096;

– MS SQL Server – 2048;

– Oracle Database – 4096;

– IBM Db2 – 2048.

Вартість за один місяць, $:

– MySQL – 1372;

– MariaDB – 1760;

– MS SQL Server – 949 (не включаючи ліцензію Enterprise);

– Oracle Database – 800 (не включаючи ліцензію Enterprise);

– IBM Db2 – 1624.

Можливість самостійного хостингу – так (1) для всіх СУБД.

Отримані дані занесено в таблицю 2.1.

Таблиця 2.1 – Показники за обраними критеріями (виконано самостійно)

СУБД	Cores	Memory (GB)	Storage (TB)	File size (MB)	Cost ($)	Self-hosting
MySQL	16	96	1	4096	1372	1
MariaDB	16	80	1	4096	1760	1

Кінець таблиці 2.1

MS SQL Server	16	64	1	2048	949	1
Oracle Database	16	32	1	4096	800	1
IBM Db2	16	64	1	2048	1624	1

Необхідно нормалізувати дані з урахуванням мінімальних та максимальних значень. За максимальний розмір файлу оцінка проставляється окремо.

Розмір нестисненого Full HD (1920x1080) файлу – приблизно 6МБ, 8K (7680x4320) – до 100МБ. Розмір стиснутого файлу залежить від параметрів стиснення. Розмір стиснутого Full HD файлу – до 0.8МБ, стиснутого 8K файлу – до 20МБ.

Кожна з СУБД підтримує великі файли до 2/4ГБ, тому всі отримують значення 1 за критерієм File Size.

В таблиці 2.2 наведені нормалізовані значення, та розраховано результат згортки за формулою 2.1.

Таблиця 2.2 – Нормалізація та результат лінійної згортки (виконано самостійно)

СУБД	Cores	Memory	Storage	File size	Cost	Self-hosting	Результат згортки
MySQL	1	1	1	1	0.4	1	0.91
MariaDB	1	0.75	1	1	0	1	0.825
MS SQL Server	1	0.5	1	1	0.84	1	0.926
Oracle Database	1	0	1	1	1	1	0.9

Кінець таблиці 2.2

IBM Db2	1	0.5	1	1	0.14	1	0.821
Ваг. коеф.	0.1	0.1	0.2	0.2	0.15	0.25	

MS SQL Server отримав найвищий результат згортки – 0.926.

Враховуючи проведений аналіз, MS SQL Server із механізмом FILESTREAM було обрано як реляційну СУБД для дослідження. Такий вибір обґрунтований його зручністю в налаштуванні, можливістю інтеграції з іншими продуктами Microsoft та високою продуктивністю при роботі з великими файлами.

2.3 Аналіз та вибір нереляційної СУБД

NoSQL бази даних розроблені для роботи з великими обсягами даних завдяки гнучкості, оптимізації та можливості масштабування. Для дослідження обрано такі альтернативи:

- MongoDB – документоорієнтована база даних, яка дозволяє зберігати зображення разом з метаданими. З використанням GridFS дозволяє зберігання файлів будь-якого розміру, розділяючи їх на чанки [23]. Також MongoDB підтримує шардінг (sharding) – метод зберігання великого обсягу даних, який розподіляє їх між кількома серверами;
- Cassandra – розподілена стовпцева база даних з високою масштабованістю та відмовостійкістю. Підтримує реплікацію – метод горизонтального масштабування, завдяки якому дані зберігаються в декількох екземплярах на різних серверах. Має високий теоретичний ліміт обсягу файлу, однак розробники цієї СУБД не рекомендують зберігати файли більші за 1МБ через негативний вплив на її продуктивність [24];
- Neo4j – графова база даних, яка може зберігати зображення з метаданими як окремі вузли. Графова модель бази даних дозволяє ефективно шукати зображення за метаданими та знаходити зв’язки між ними. Також нативно підтримує деякі методи машинного навчання, такі як класифікація, прогнозування зв’язків [25];

- Azure Cosmos DB – мультимодельна хмарна база даних від Microsoft, яка підтримує різні моделі даних (JSON документи, графова модель, ключ-значення, стовпцева модель, реляційна модель) та різні API (NoSQL, MongoDB, PostgreSQL, Cassandra, Gremlin, Table), що надає можливість працювати з іншими базами даних, використовуючи платформу Azure та її переваги, такі як глобальна розподіленість, та SLA (Service Level Agreement – домовленість рівня послуг) на рівні 99.999% [26];
- PostgreSQL – реляційна СУБД, було додано її до нереляційних через гібридний підхід між реляційними та нереляційними СУБД, підтримку формату JSONB (JSON Binary), який дозволяє зберігати неструктуровані дані подібні до NoSQL рішень, але в бінарному вигляді, що в свою чергу дозволяє підвищити ефективність читання та обробки даних [27] .

Були обрані наступні хостинги з пакетами від 1ТБ сховища для отримання значень за деякими критеріями, коли база даних хоститься не самостійно:

- MongoDB – MongoDB Dedicated M140 Google Cloud [5];
- Cassandra – OVHcloud Enterprise DB1-15 for Cassandra [28];
- Neo4j – AuraDB Business Critical Pricing 1024GB Storage [29];
- Azure Cosmos DB – Azure Cosmos DB Calculator, 1000GB storage, item size upto 2048KB [30];
- PostgreSQL – OVHcloud Enterprise DB1-30 for PostgreSQL [28].

Кількість виділених процесорів:

- MongoDB – 48;
- Cassandra – 24;
- Neo4j – 96;
- Azure Cosmos DB – автомасштабування в залежності від Request Units/s (одиниця запиту за секунду. 1 одиниця запиту = повернення об’єкту розміром 1КБ); надано найвище значення серед альтернатив (96);
- PostgreSQL – 24.

Обсяг виділеної оперативної пам’яті, ГБ:

- MongoDB – 192;
- Cassandra – 90;
- Neo4j – 512;
- Azure Cosmos DB – автомасштабування; надано найвище значення серед альтернатив (512);
- PostgreSQL – 90.

Максимальний розмір одного сховища, ТБ:

- MongoDB – 1;
- Cassandra – 5.76;
- Neo4j – 1;
- Azure Cosmos DB – 1;
- PostgreSQL – 1.92.

Максимальний розмір файлу, МБ:

- MongoDB – 16, необмежений з використанням GridFS – технології, що поділяє файли на чанки;
- Cassandra – теоретичний максимум 2048, на практиці розробники рекомендують зберігати файли лише до 1;
- Neo4j – теоретичний максимум 2048;
- Azure Cosmos DB – 2;
- PostgreSQL – 1024.

Вартість за один місяць:

- MongoDB – $6,868.8;
- Cassandra – $1,653.72;
- Neo4j – $74,752;
- Azure Cosmos DB – $3,294.08;
- PostgreSQL – $1,587.72.

Можливість самостійного хостингу:

- MongoDB – так (1);
- Cassandra – так (1);

- Neo4j – так (1);
- Azure Cosmos DB – ні (0);
- PostgreSQL – так (1).

Отримані значення було занесено в таблицю 2.3.

Таблиця 2.3 – Отримані значення за критеріями (виконано самостійно)

СУБД	Cores	Memory (GB)	Storage (TB)	File size (MB)	Cost ($)	Self-hosting
MongoDB	48	192	1	16	6,868	1
Cassandra	24	90	5.76	1	1,653	1
Neo4j	96	512	1	2048	74,752	1
Azure	96	512	1	2	3,294	0
PostgreSQL	24	90	1.92	1024	1,587	1

Аналогічно до аналізу реляційних СУБД необхідно нормалізувати показники. Neo4j, PostgreSQL та MongoDB (з використанням GridFS) підтримують великі файли (1), Azure Cosmos DB та Cassandra підтримують (рекомендують зберігати) лише файли невеликого розміру (0).

Нижче наведено нормалізовані з урахуванням мінімальних та максимальних значень дані та результати згортки за формулою 2.1 (див. табл. 2.4).

Таблиця 2.4 – Нормалізація та розрахування результату згортки (виконано самостійно)

СУБД	Cores	Memory	Storage	File size	Cost	Self-hosting	Результат згортки
MongoDB	0.333	0.242	0	1	0.928	1	0.647
Cassandra	0	0	1	0	0.999	1	0.6
Neo4j	1	1	0	1	0	1	0.65
Azure	1	1	0	0	0.977	0	0.347

Кінець таблиці 2.4

PostgreSQL	0	0	0.193	1	1	1	0.639
Ваг. коеф.	0.1	0.1	0.2	0.2	0.15	0.25	

Нижче наведені альтернативи відсортовані за корисністю:

– Neo4j (0.65);
– MongoDB (0.647);
– PostgreSQL (0.639);
– Cassandra (0.6);
– Azure Cosmos DB (0.347).

На основі проведеного аналізу та порівняння п’яти альтернативних нереляційних СУБД, було обрано MongoDB для дослідження методів зберігання великого обсягу візуальних даних. Вибір обґрунтовано її продуктивністю, гнучкістю у роботі з великими обсягами даних завдяки GridFS, підтримці горизонтального та вертикального масштабування, а також можливості інтеграції з популярними фреймворками та сучасними технологіями машинного навчання.

2.3 Аналіз та вибір предметної області дослідження

Є безліч випадків, в яких присутня велика кількість візуальної інформації:

– соціальні мережі – поширені зображення та відео;
– медицина – результати різних досліджень;
– різні бази знань (наприклад, більше ніж 50% англомовних статей у Вікіпедії містять хоча б одне зображення [31]), мапи, супутникові знімки;
– набори даних для машинного навчання.

Було обрано соціальні мережі як предметну область для дослідження.

З кожним роком зростає кількість користувачів інтернету. За 2022 по 2024 рік кількість користувачів збільшилась на 500 мільйонів людей, з 5 мільярдів до 5.5, або з 63% до 68% популяції землі. І більш ніж 90% з онлайн користувачів – користувачі соціальних мереж [1, 2].

Візуальна інформація дуже розповсюджена та різноманітна в соціальних мережах. Починаючи від маленьких зображень профіля користувача, закінчуючи публікаціями з безліччю фотографій високої якості. І з кожним днем кількість цієї інформації зростає. Знаходження оптимального методу зберігання цієї інформації покращить досвід кожного з користувачів соціальних мереж.

Змодельовану мінімалізовану ER-діаграму предметної області, яка містить наступні сутності (див. рис. 2.1):

– користувач – власне користувач з властивостями ім’я та зображенням в профілі;
– публікація – публікації в соціальній мережі, містить інформацію про автора та дату, текст та ідентифікатор іншої публікації, для якої ця є коментарем. Тобто, допускається рекурсивний зв’язок один до багатьох (в однієї публікації може бути 0-безліч коментарів);
– зображення – метадані та зміст зображення, посилання на ідентифікатор публікації.

Зв’язок між сутностями «Користувач»-«Публікація» та «Публікація»-«Зображення» – один до багатьох.

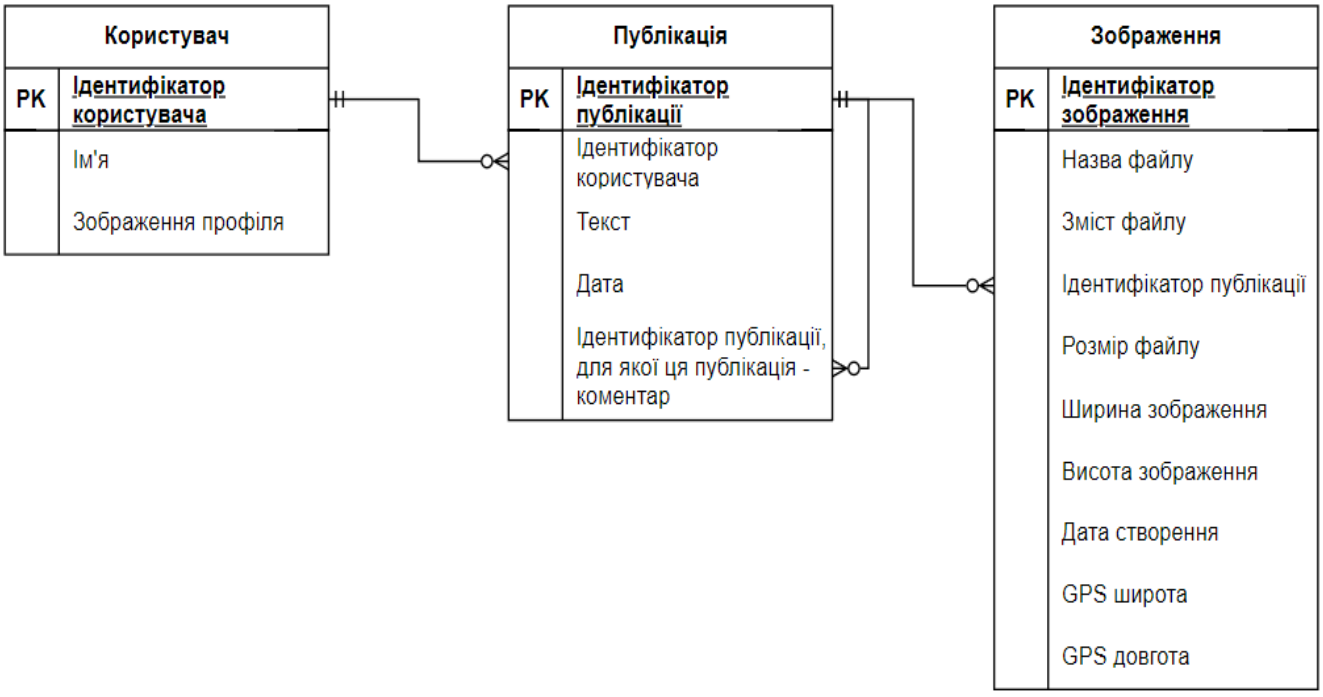

Рисунок 2.1 – ER-діаграма предметної області (рисунок виконаний самостійно)

В реляційному підході зображення зберігається в окремій від публікації таблиці Image разом з метаданими. Для використання FILESTREAM необхідно буде просто додати цей атрибут при створенні таблиці. При цьому ніяких змін до схеми бази даних не буде (див. рис. 2.2).

В нереляційному підході завдяки можливості вкладання документів зображення будуть міститися в масиві в документі сутності, метадані – теж окремий документ, який вкладено в документ зображення. Вкладання метаданих в зображення та зображення в публікацію на схемі наведено як зв'язок між сутностями. (див. рис. 2.3)

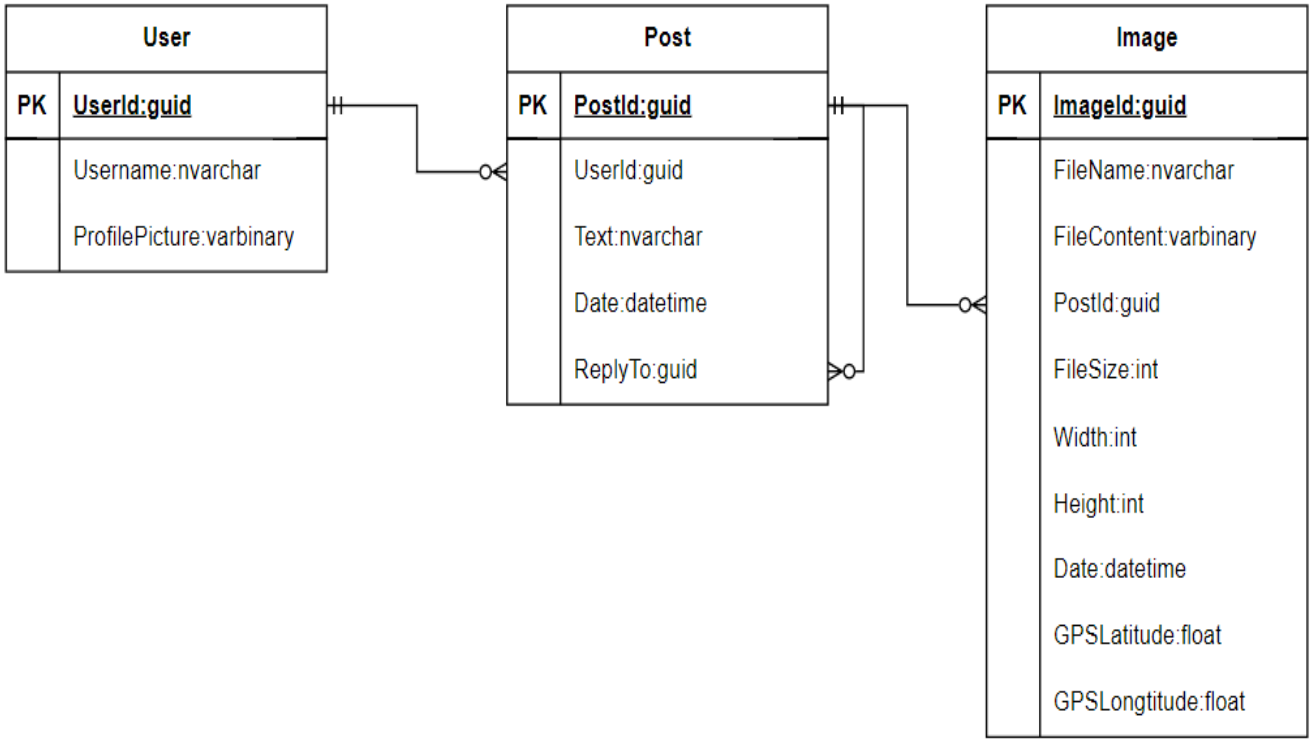

Рисунок 2.2 – Схема бази даних для MS SQL Server (рисунок виконаний самостійно)

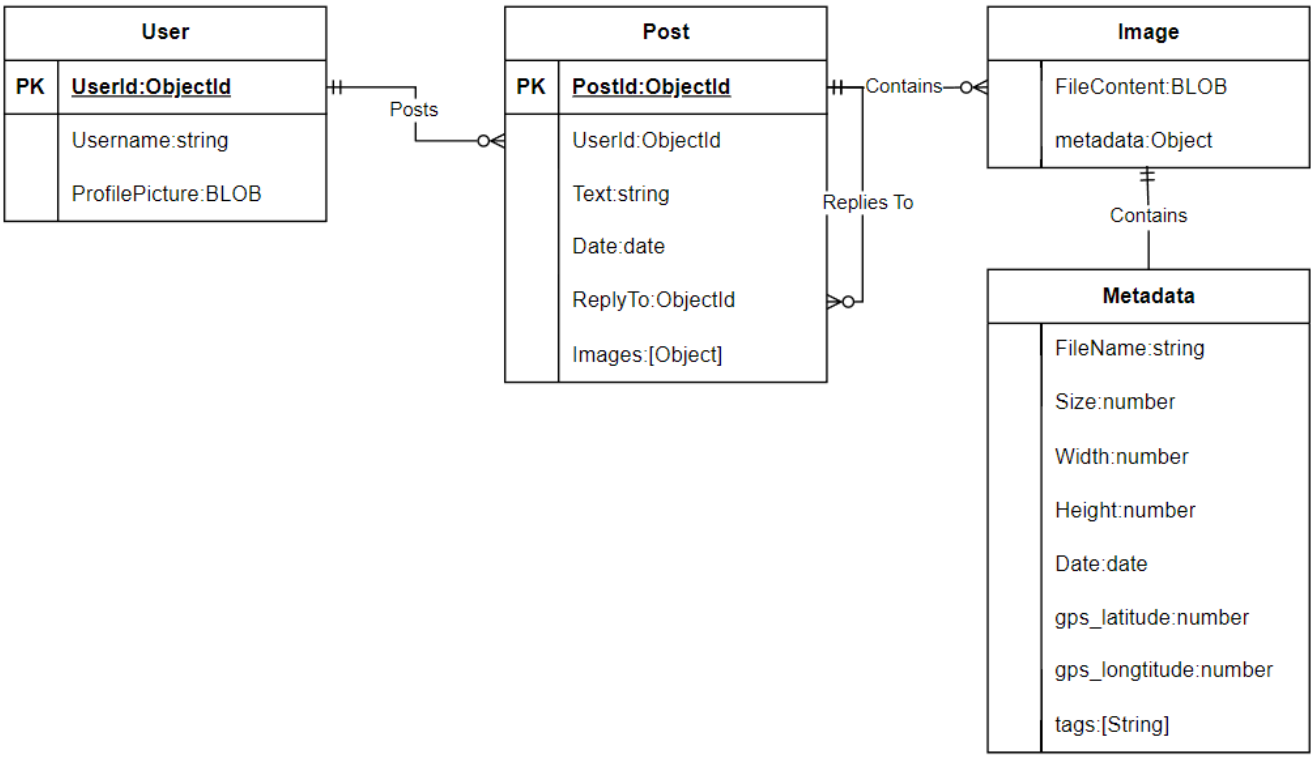

Рисунок 2.3 – Схема бази даних для MongoDB (рисунок виконаний самостійно)

Для підходу з гібридним сховищем саме зображення зберігається на окремому сховищі, а в базі даних міститься посилання на файл та метадані (див. рис. 2.4).

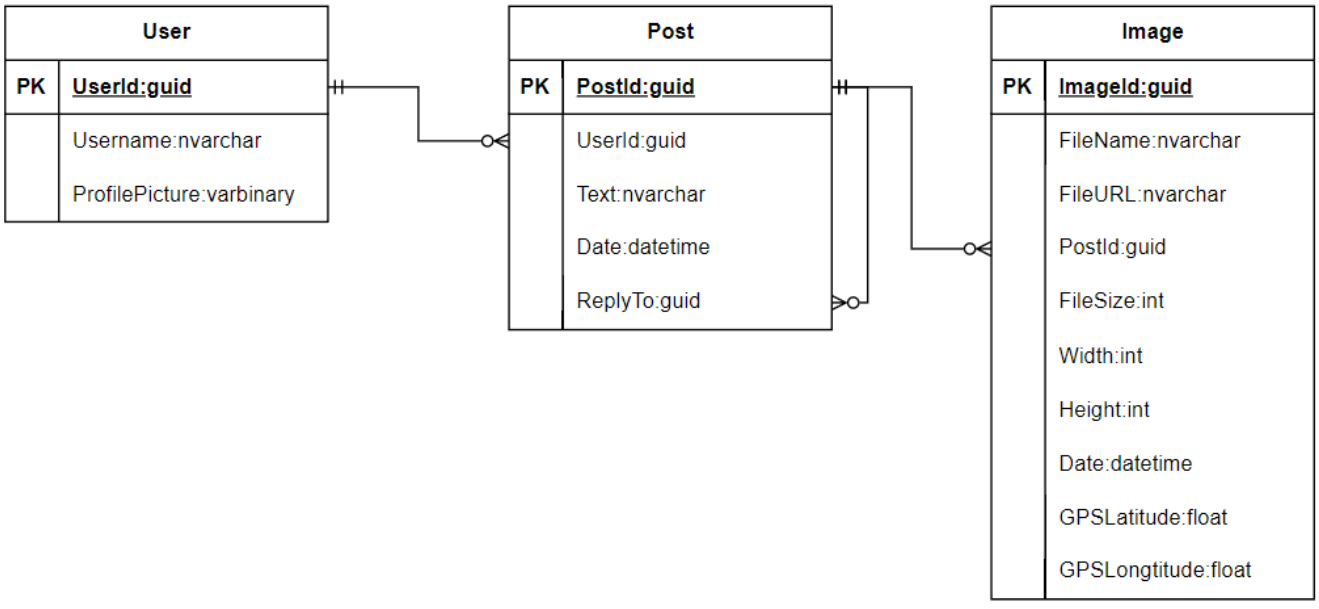

Рисунок 2.4 – Схема бази даних для MS SQL Server з гібридним сховищем (рисунок виконаний самостійно)

Пошук за метаданими в реляційному варіанті повинен бути швидким, тому що це доступ до однієї таблиці. З іншого боку в нереляційному варіанті метадані знаходяться в документі на другому рівні вкладеності. Але завдяки можливості індексації записів за будь-якими полями, пошук за метаданими в нереляційній СУБД також повинен бути достатньо швидким.

Запропонованих сутностей повинно бути достатньо для проведення експериментального дослідження та отримання результатів про ефективність того чи іншого методу зберігання зображень в базах даних.

2.4 Планування експериментального дослідження

З метою економії трафіку та дискового простору, виконується стиснення при завантаженні зображень до соціальних мереж. Отже, вхідною інформацією будуть JPEG зображення Full HD (1920x1080) розміром до 0.8МБ, та 8K UHD (7680 × 4320) – до 20МБ.

Для визначення ефективності методів зберігання графічної інформації в базах даних, необхідно перевірити запис, читання та фільтрацію даних для кожного з підходів.

Буде перевірено наступні методи:

- MS SQL Server без використання FILESTREAM для зберігання зображень;
- MS SQL Server з використанням FILESTREAM;
- MS SQL Server з посиланнями на зображення в хмарному сховищі;
- MongoDB з вкладенням зображень;
- MongoDB з вкладенням посилань на зображення в хмарному сховищі.

Критеріями успішності для порівняння методів зберігання графічної інформації обрано:

- час виконання операцій (запису, читання, фільтрації), що дозволяє оцінити продуктивність кожного підходу;
- зайнятий обсяг дискового простору, який демонструє ефективність зберігання даних;
- рівень використання системних ресурсів (оперативна пам’ять, процесор).

На основі цих трьох критеріїв можна буде зробити висновок про масштабованість кожного підходу, що є критично важливим для роботи із соціальними мережами, де кількість даних постійно зростає.

Для кожного з методів зберігання буде виконано запис різних обсягів даних (1000 FullHD зображень, 10000 FullHD, 1000 8K UHD, 10000 8K UHD), їхнього читання, та фільтрацію за метаданими.

Для цих тестів будуть отримуватися наступні значення, які зберігатимуться в наступному вигляді (див. рис 2.5):

– час додавання;
– час читання;
– час фільтрації;
– використана оперативна пам'ять;
– використане локальне сховище.

	1000HD		10000HD		1000UHD		10000UHD	
	Time (ms)	Memory (MB)	Time (ms)	Memory (MB)	Time (ms)	Memory (MB)	Time (ms)	Memory (MB)
InsertOne								
ReadAll								
FilterOne								
UserSize								
PublicationSize								
TotalSize								

Рисунок 2.5 – Структура результатів тестів (рисунок виконаний самостійно)

Результати тестів із різними обсягами даних та розмірами зображень (FullHD, 8K UHD) дозволять оцінити переваги та недоліки кожного з методів, зокрема використання FILESTREAM у MS SQL Server для файлів різного розміру, використання MongoDB або гібридного сховища. Такий підхід сприяє оптимізації роботи із зображеннями, враховуючи вимоги до економії трафіку, дискового простору та продуктивності системи.

Таким чином, заплановане експериментальне дослідження має на меті порівняти продуктивність методів зберігання графічної інформації у реляційних та

нереляційних базах даних з урахуванням різних обсягів та форматів даних. Очікується, що отримані результати допоможуть розробникам програмного забезпечення оптимізувати вибір архітектури бази даних залежно від конкретних потреб, таких як масштабованість, швидкість доступу до даних та економічна ефективність. Висновки дослідження також будуть корисними для вдосконалення існуючих практик зберігання візуальних даних у таких сферах, як соціальні мережі, медицина та автоматизовані системи.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ ТА РЕАЛІЗАЦІЯ

3.1 Проектування програмного застосунку для виконання експериментального дослідження

Від проектування та реалізації застосунку залежить точність результатів дослідження. Різниця апаратного забезпечення хостнгів реляційних та нереляційних баз даних може вплинути на об’єктивність отриманих результатів.

Саме тому бази даних буде розміщено на локальному комп’ютері, також буде використано локальне файлове сховище. На рисунку 3.1 наведено діаграму розміщення програмної системи.

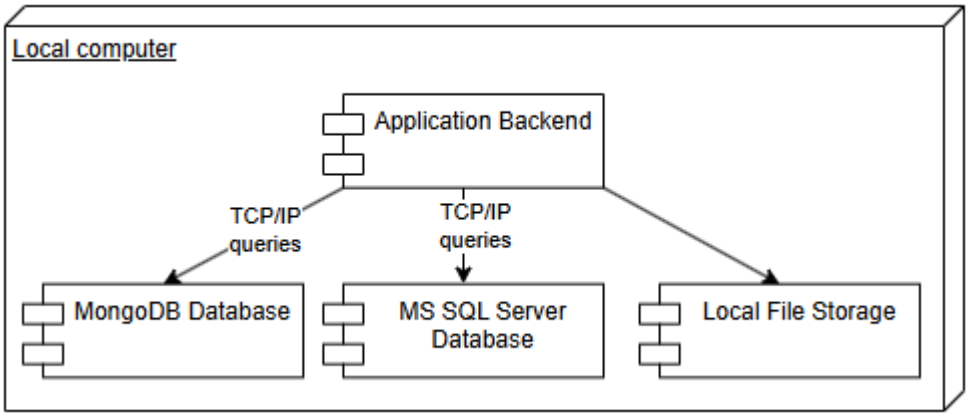

Рисунок 3.1 – Діаграма розміщення програмної системи для експериментального дослідження (рисунок виконаний самостійно)

Сам програмний застосунок буде складатися з трьох логічно розподілених шарів:

- WebLayer – веб-шар, який приймає HTTP запити;
- BusinessLayer – шар, на якому міститься логіка застосунку. У випадку цього дослідження – операції роботи з даними та вимірювання продуктивності застосунку.
- DataLayer – шар взаємодії з базою даних, саме він міститиме CRUD (Create, Read, Update, Delete) операції.

Розподілення застосунку на логічні шари надає такі переваги, як чітке розділення обов’язків, масштабованість системи, гнучкість та безпека. Такий підход наближає серверну частину застосунку до реальних працюючих систем.

3.2 Вибір технологій для програмної реалізації

Для реалізації серверної частини обрано технологію ASP.NET CORE [32] – високопродуктивний фреймворк від Microsoft для розробки веб-застосунків.

Для роботи з базами даних буде використано ORM (object-relational mapping) фреймворк EntityFrameworkCore [33], який дозволяє використання підходу code-first при розробці бази даних. Наявність надбудов для роботи як з MS SQL Server, так і з MongoDB, дозволяє розробляти код для різних конфігурацій програмної системи з мінімальними змінами до коду.

Взаємодію з серверною частиною буде виконано через інструмент Swagger/OpenAPI [34], який дозволяє автоматично генерувати специфікацію веб-інтерфейсу серверної частини і взаємодіяти з нею.

Для конфігурацій баз даних зі зберіганням зображень на окремому сховищі було обрано хмарне сховище на Microsoft Azure.

Вимірювання швидкодії серверної частини застосунку буде виконано з використанням бібліотеки для бенчмаркінгу BenchmarkDotNet [35].

3.3 Розробка серверної частини програмної системи

Як було зазначено раніше, серверна частина програмної системи складатиметься з трьох логічно розділених шарів.

DataLayer відповідає за взаємодію серверної частини з базою даних.

Спочатку необхідно розробити моделі об’єктів предметної області та контекст бази даних. Нижче наведено код моделі публікації, яка містить в собі посилання на автора публікації та на список зображень, прикладених до публікації, що відповідає схемі бази даних з діаграми на рисунку 3.1.

```
namespace DbTest1.DataLayer.Models
{
    public class Publication
    {
        public Publication()
        {
            Images = new List<Image>();
        }

        public Guid Id { get; set; }
        public Guid UserId { get; set; }
        public User User { get; set; }
        public string Text { get; set; }
        public DateTime Date { get; set; }
        public Guid? ReplyToId { get; set; }
        public Publication? ReplyTo { get; set; }
        public IEnumerable<Publication> Replies { get; set; }
        public IEnumerable<Image> Images { get; set; }
    }
}
```

Аналогічно прикладу вище, створено власне моделі профілю користувача та зображення. Далі ці моделі треба поєднати в контекст бази даних. Нижче наведено приклад створення контексту бази даних, який містить 3 таблиці (DbSet<>): користувачі, публікації та зображення. В методі OnModelCreating() з використанням Fluent API явно наведені зв’язки між таблицями бази даних.

```
using DbTest1.DataLayer.Models;
using Microsoft.EntityFrameworkCore;

namespace DbTest1.DataLayer
{
    public class DbTestDbContext : DbContext
    {
        public DbTestDbContext(DbContextOptions<DbTestDbContext>
options) : base(options)
        { }

        public DbSet<User> User { get; set; }
        public DbSet<Publication> Publication { get; set; }
        public DbSet<Image> Image { get; set; }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.Entity<User>()
                .HasMany(u => u.Publications)
                .WithOne(p => p.User)
                .HasForeignKey(p => p.UserId);

            modelBuilder.Entity<Publication>()
                .HasMany(p => p.Replies)
```

```
                .WithOne(r => r.ReplyTo)
                .HasForeignKey(p => p.ReplyToId);

            modelBuilder.Entity<Publication>()
                .HasMany(p => p.Images)
                .WithOne(i => i.Publication)
                .HasForeignKey(i => i.PublicationId);
        }
    }
}
```

Цього достатньо, щоб створити «міграції» – код переходу між станами таблиць баз даних. Виконання команди Add-Migration Initial створить міграцію «Initial», в якій буде міститись код створення всіх таблиць бази даних. На рисунку 3.2 наведено приклад виклику цієї команди та фрагмент створеного коду міграції.

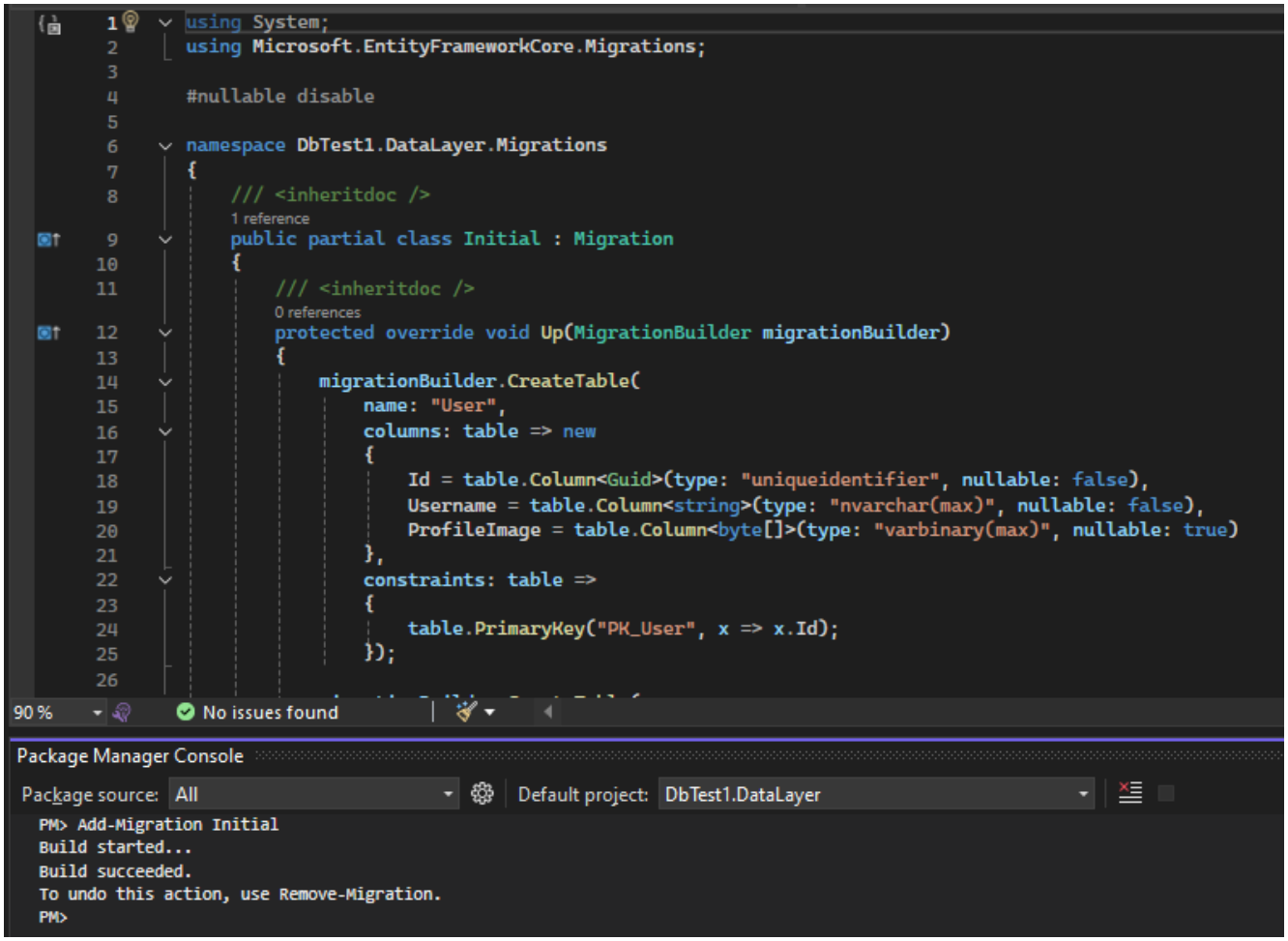

Рисунок 3.2 – Перша «міграція» бази даних (рисунок виконаний самостійно)

Виклик команди Update-Database <Migration> виконує код міграцій для переходу до останньої чи специфічно обраної міграції. Після виклику команди

було створено таблиці бази даних (див. рис. 3.3), які відповідають контексту бази даних, наведеному вище.

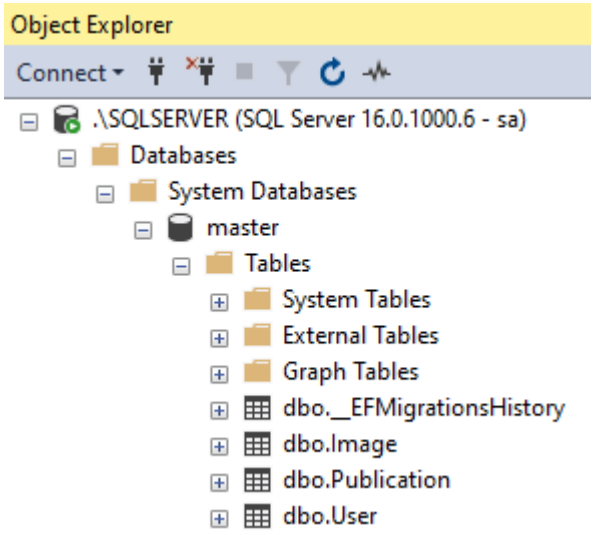

Рисунок 3.3 – Створені таблиці (рисунок виконаний самостійно)

Для взаємодії з даними використовуються так звані репозиторії. Нижче наведено код параметризованого класу репозиторію.

```
using DbTest1.Repositories.Repository;
using Microsoft.EntityFrameworkCore;

namespace DbTest1.DataLayer.Repositories.Repository
{
    public class Repository<TEntity> : IRepository<TEntity> where
TEntity : class
    {
        protected readonly DbTestDbContext Context;
        public Repository(DbTestDbContext context)
        {
            Context = context;
        }
        public virtual async Task<TEntity> Get(Guid id)
        {
            return await Context.Set<TEntity>().FindAsync(id);
        }

        public async Task<IEnumerable<TEntity>> GetAll()
        {
            return await Context.Set<TEntity>().ToListAsync();
        }

        public async Task<TEntity> Insert(TEntity entity)
        {
            await Context.Set<TEntity>().AddAsync(entity);
```

```
                return entity;
            }

            public void Delete(TEntity entity)
            {
                Context.Set<TEntity>().Remove(entity);
            }

            public TEntity Update(TEntity entity)
            {
                Context.Set<TEntity>().Update(entity);
                return entity;
            }

            public async Task Save()
            {
                await Context.SaveChangesAsync();
            }

            public async Task Truncate()
            {
                await Context.Database.ExecuteSqlRawAsync("TRUNCATE TABLE
[TableName]");
            }
        }
    }
```

Цей клас містить всі необхідні CRUD операції для роботи з базою даних. Команди використовуються на таблиці Context.Set<TEntity>, де TEntity – модель об’єкту бази даних.

Для створення репозиторію для конкретної таблиці достатньо лише наслідувати клас репозиторію з необхідним параметром.

```
    using DbTest1.DataLayer.Models;
    using DbTest1.DataLayer.Repositories.Repository;

    namespace DbTest1.DataLayer.Repositories.UserRepository
    {
        public class UserRepository : Repository<User>, IUserRepository
        {
            public UserRepository(DbTestDbContext context) :
base(context) { }
        }
    }
```

Використання універсального репозиторію дозволяє уникнути написання однакового коду для всіх таблиць бази даних. Таким чином класи конкретних репозиторіїв містять лише код, специфічний для цієї таблиці.

BusinessLayer серверної частини містить власне логіку системи, тобто вимірювання продуктивності роботи системи. Нижче код перевірки продуктивності на прикладі вимірювання ефективності запису даних.

```
using BenchmarkDotNet.Attributes;
using DbTest1.DataLayer;
using DbTest1.DataLayer.Models;
using DbTest1.DataLayer.Repositories.ImageRepository;
using DbTest1.DataLayer.Repositories.PublicationRepository;
using DbTest1.DataLayer.Repositories.UserRepository;
using Microsoft.EntityFrameworkCore;

namespace DbTest1.BusinessLayer.Benchmarks.SQLServerLocalBenchmark
{
    [MemoryDiagnoser]
    [MinIterationCount(1)]
    [MaxIterationCount(100)]
    [InvocationCount(20)]
    public class SQLServerLocalBenchmark
    {
        private  IUserRepository _userRepository;
        private  IPublicationRepository _publicationRepository;
        private  IImageRepository _imageRepository;
        private int iter = 0;
        private readonly int size = 800000;

        public SQLServerLocalBenchmark(){}

        [GlobalSetup]
        public void Setup()
        {
            var options = new
DbContextOptionsBuilder<DbTestDbContext>()
            .UseSqlServer(connectionString)
            .Options;

            var _context = new DbTestDbContext(options);
            _context.Database.EnsureCreated();
            _userRepository = new UserRepository(_context);
            _publicationRepository = new
PublicationRepository(_context);
            _imageRepository = new ImageRepository(_context);
        }


        [Benchmark]
        public async Task InsertBenchmark()
        {
            Random random = new Random();

            User user = new User()
            {
                Id = Guid.NewGuid(),
                Username = "Test",
```

```
                    ProfileImage = new byte[1024]
                };
                user = await _userRepository.Insert(user);

                Publication publication = new Publication()
                {
                    Id = Guid.NewGuid(),
                    UserId = user.Id,
                    Text = $"Test {iter + 1}",
                    Date = DateTime.Now
                };
                publication = await
_publicationRepository.Insert(publication);

                long filesize = random.Next(size / 2) + size / 2;
                Image image = new Image()
                {
                    Id = Guid.NewGuid(),
                    PublicationId = publication.Id,
                    FileName = $"image{iter + 1}.jpg",
                    FileContent = new byte[filesize],
                    FileSize = filesize,
                    Width = 1920,
                    Height = 1080,
                    Date = DateTime.Now
                };
                image = await _imageRepository.Insert(image);
                ++iter;
                await _imageRepository.Save();
            }
        }
    }
```

В методі Setup виконується налаштування класу, приєднання до бази даних. Атрибут [Benchmark] вказує на методи, продуктивність яких необхідно виміряти. В даному прикладі це метод InsertBenchmark().

Налаштування параметрів бенчмаркінгу (кількості ітерацій, розмір зображення) виконуватиметься через зміну параметрів класу, продуктивність якого досліджується.

Шар контролерів виконує роль API серверної частини застосунку. Спочатку треба провести налаштування застосунку – підключення до бази даних, створення сервісів, екземплярів класів для Dependency Injection, додавання необхідних бібліотек тощо. Нижче наведено код програми.

```
using DbTest1.DataLayer;
using DbTest1.DataLayer.Repositories.ImageRepository;
using DbTest1.DataLayer.Repositories.PublicationRepository;
```

```
using DbTest1.DataLayer.Repositories.UserRepository;
using Microsoft.EntityFrameworkCore;
internal class Program
{
    private static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.

        builder.Services.AddControllers();
        builder.Services.AddDbContext<DbTestDbContext>(opt =>
        {
            opt.UseSqlServer(connectionString);
        });

        builder.Services.AddTransient<IImageRepository,
ImageRepository>();
        builder.Services.AddTransient<IPublicationRepository,
PublicationRepository>();
        builder.Services.AddTransient<IUserRepository,
UserRepository>();

        builder.Services.AddSwaggerGen(options =>
        {
            options.SwaggerDoc("v1", new
Microsoft.OpenApi.Models.OpenApiInfo());
        });

        var app = builder.Build();

        // Configure the HTTP request pipeline.

        app.UseHttpsRedirection();

        app.UseAuthorization();

        app.UseSwagger();

        app.MapControllers();

        app.UseSwaggerUI();

        app.Run();
    }
}
```

Нижче наведено приклад коду контролера виміру продуктивності запису.

```
using BenchmarkDotNet.Running;
using DbTest1.BusinessLayer.Benchmarks.SQLServerLocalBenchmark;
using Microsoft.AspNetCore.Mvc;

namespace DbTest1.Controllers
{
    [ApiController]
```

```
        [Route("api/[controller]")]
        public class InsertBenchmarkController : Controller
        {
            public InsertBenchmarkController()
            {
            }

            [HttpGet]
            [Route("benchmarkInsert")]
            public async Task<IActionResult> Insert()
            {
                BenchmarkRunner.Run<SQLServerLocalBenchmark>();
                return Ok();
            }

        }
    }
```

Використання бібліотеки Swagger дозволяє автоматично генерувати API специфікацію та взаємодіяти з нею через веб-браузер. На рисунку 3.4 наведено зображення інтерфейсу Swagger після виклику одного з контролерів.

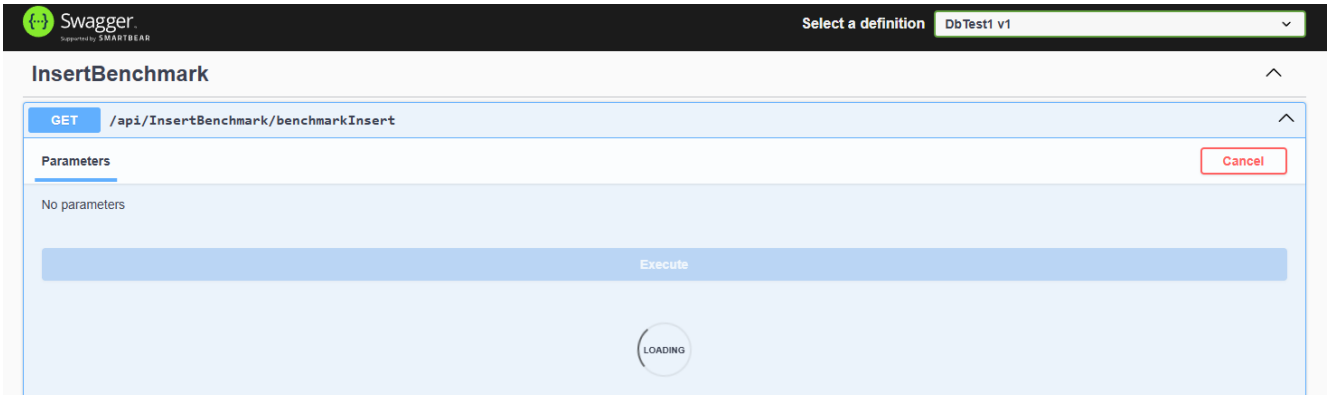

Рисунок 3.4 – Інтерфейс Swagger (рисунок виконаний самостійно)

Після виконання цього виклику в консоль виводяться результати вимірів продуктивності роботи програми (див. рис. 3.5).

```
InvocationCount=20  MaxIterationCount=100  MinIterationCount=1
UnrollFactor=1

| Method          | Mean     | Error    | StdDev   | Gen0     | Allocated |
|---------------- |---------:|---------:|---------:|---------:|----------:|
| InsertBenchmark | 11.48 ms | 0.787 ms | 2.271 ms | 150.0000 |   5.34 MB |
```

Рисунок 3.5 – Результати вимірів (рисунок виконаний самостійно)

На прикладі показано, було виконано 2000 записів користувача та публікації з зображенням. Одна операція триває в середньому 11.48 мілісекунд.

Аналогічно реалізовано бенчмарки і кінцеві точки контролерів для інших операцій: читання, фільтрація (пошук).

Розроблений застосунок гнучкий завдяки використанню платформи ASP.NET Core, тому для реалізації змін до конфігурації програмної системи чи бази даних достатньо змін декілької строк коду.

Таким чином, реалізовано програмну систему для тестування продуктивності різних методів зберігання великих обсягів візуальних даних в реляційних та нереляційних системах, якої повинно бути достатньо для проведення об’єктивнів експериментів та отримання висновків про продуктивність тих чи інших конфігурацій. Програмний код розміщено в GitHub репозиторії vitalii-nikolenko/DbTest [36].

4 ОПИС ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Проведення експериментальних досліджень

Після планування експериментального дослідження та програмної реалізації застосунку можна безпосередньо перейти до виконання досліджень.

Очікуються наступні результати:

- Microsoft SQL Server з використанням FILESTREAM виявиться найшвидшим за рахунок окремого зберігання файлів від таблиць баз даних;
- Microsoft SQL Server зі зберіганням файлів на окремому сховищі буде повільнішим за використання FILESTREAM через необхідність реалізації окремого доступу до файлів. Покаже кращий результат ніж аналогічний підхід з MongoDB через швидкодію роботи зі структурованими даними;
- MongoDB з окремим сховищем;
- MongoDB зі зберіганням зображень в базі даних буде краще ніж MS SQL Server з аналогічним підходом через оптимізацію нереляційних СКБД для роботи з неструктурованими даними;
- Microsoft SQL Server зі зберіганням зображень в базі даних покаже найгірший результат через вплив фрагментації на сторінки пам’яті.

Для тестів використано ноутбук з процесором AMD Ryzen 5 5500u з 6 фізичними ядрами, 12 потоками. Базова частота: 2.1 ГГц, частота Boost: 4 ГГц. 16ГБ DDR4 3200 МГц оперативної пам’яті.

Технології використані для розробки та підтримки програмної системи та виконання експерименту:

- система Microsoft Windows 10 22H2;
- .NET Core 9.0.201, C# 13.0;
- середа розробки Microsoft Visual Studio 2022;
- СКБД Microsoft SQL Server 2022 та MongoDB;
- SQL Server Management Studio;
- MongoDB Atlas;

- ORM (Object-Relational Mapping) Microsoft Entity Framework Core;
- OpenAPI Swagger UI – інтерфейс для документації, тестування та взаємодії з API застосунку;
- BenchmarkDotNet – фреймворк для виконання тестів продуктивності програмного застосунку.

Фреймворк для бенчмаркінгу автоматично знаходить та видаляє викиди – значення тестів, що значно відрізняються відрізняються від решти результатів. Операції читання та запису виконуються порціями по 20 записів. Загалом виконується 1000 чи 10000 операції запису в залежності від експерименту. Для читання та фільтрації операції виконуються повторно, допоки середній час виконання не стабілізується, чи не буде досягнуто ліміт кількості операцій.

Після цього розраховується середній час кожної з операції.

$$T(1) = \frac{T(N)}{N} \tag{4.1}$$

де *T(1)* – середній час виконання однієї операції,

T(N) – загальний час виконання,

N – кількість операцій.

В таблиці 4.1 наведено час виконання запису інформації в базу даних для всіх методів зберігання даних та обсягів даних. Наведені результати є середнім часом однієї операції в мілісекундах.

Таблиця 4.1 – Середній час операції запису даних (таблиця виконана самостійно)

	Обсяг даних			
Метод зберігання	1000FHD	10000FHD	1000UHD	10000UHD
MSSQLLocal	9.99	22.42	51.86	84.94

Кінець таблиці 4.1

MSSQLFilestream	9.98	10.69	47.25	179.01
MSSQLFilestorage	3.21	2.86	20.41	19.38
MongoDBLocal	15.13	16.66	173.50	175.90
MongoDBFilestorage	3.68	3.20	13.98	20.88

Перелік позначень:

- 1000FHD, 10000FHD, 1000UHD, 10000UHD – експерименти з 1000 та 10000 записів FullHD та UHD відповідно;
- MSSQLLocal – підхід з локальним зберіганням зображень в базі даних Microsoft SQL Server;
- MSSQLFilestream – Microsoft SQL Server з використанням технології FILESTREAM;
- MSSQLFilestorage – Microsoft SQL Server зі зберіганням зображень в окремому сховищі;
- MongoDBLocal – підхід з локальним зберіганням зображень в базі даних MongoDB;
- MongoDBFilestorage – MongoDB зі зберіганням зображень в окремому сховищі.

В таблиці 4.2 наведено середній час однієї операції послідовного читання даних.

Таблиця 4.2 – Середній час операції читання даних (таблиця виконана самостійно)

	Обсяг даних			
Метод зберігання	1000FHD	10000FHD	1000UHD	10000UHD
MSSQLLocal	2.737	2.922	1369.512	1462.0804
MSSQLFilestream	3.925	4.8206	2599.218	3192.3036

Кінець таблиці 4.2

MSSQLFilestorage	0.4782	1.983	0.503	4.712
MongoDBLocal	0.8455	1.392	18.62	305
MongoDBFilestorage	0.5905	1.943	7.86	18.84

В таблиці 4.3 наведено середній час однієї операції фільтрації (пошуку) даних.

Таблиця 4.3 – Середній час операції фільтрації даних (таблиця виконана самостійно)

	Обсяг даних			
Метод зберігання	1000FHD	10000FHD	1000UHD	10000UHD
MSSQLLocal	7.746	9.354	1472	1502
MSSQLFilestream	5.441	6.892	1569	1987.4192
MSSQLFilestorage	1.506	3.592	13.47	17.76
MongoDBLocal	114.2	1968	3748	52224
MongoDBFilestorage	5.063	27.68	13.13	41.76

З отриманих результатів можна побачити, що, всупереч очікуванням, продуктивність з використанням Microsoft SQL Server FILESTREAM виявилась найгіршою. Відсутність кешування файлів в буферному пулі бази даних збільшила кількість звернень до файлової системи операційної системи, що разом з загальними накладними витратами на зберігання файлів знизило продуктивність серверу бази даних.

Для підходу MSSQLLocal помітно значне знижчення продуктивності зі збільшенням обсягу та розміру файлів. В буферний пул Microsoft SQL Server, що знаходиться в оперативній пам’яті, завантажуються дані, до яких зверталася база даних. Якщо розмір бази даних більший за обсяг вільної оперативної пам’яті, збільшується частота операцій вивільнення пам’яті та кількість звернень до

жорсткого диску для дозавантаження нових сторінок пам’яті. Це єдиний метод, з яким використання оперативної пам’яті системи досягло 100%.

Хоч MongoDB і оптимізована для неструктурованих даних, робота з великими бінарними даними, як і в випадку з MSSQLLocal, значно зменшує продуктивність роботи бази даних.

Отже, найкращими за продуктивністю виявились методи з гібридним сховищем MSSQLFilestorage та MongoDBFilestorage. Завдяки відсутності фрагментації даних та низькому розміру самих даних, можливість завантаження всієї бази даних в оперативну пам’ять значно підвищила продуктивність роботи.

4.2 Оцінка масштабованості методів зберігання даних

Масштабованість – здатність системи справлятися зі збільшенням навантаження. Чим нижче втрата продуктивності методу зберігання даних, тим краще система працює під навантаженням. Для кожного з експериментів було розраховано втрату продуктивності (збільшення часу виконання).

$$\eta = \frac{T_1}{T_2}, \tag{4.2}$$

де η – втрата продуктивності,

T_1 та T_2 середній час виконання експериментів.

Розраховано втрату продуктивності запису (див. табл. 4.4), читання (див. табл. 4.5) та фільтрації (див. табл. 4.6) для наступних змін обсягу даних:

- 1000FHD to 10000FHD – збільшення кількості записів FullHD файлів;
- 1000FHD to 1000UHD – збільшення розміру зображення з FullHD до 8K UltraHD;
- 1000UHD to 10000UHD – збільшення кількості записів 8K UltraHD файлів;
- 1000FHD to 10000UHD – загальна втрата продуктивності зі збільшенням розміру та кількості файлів.

Таблиця 4.4 – Втрата продуктивності запису даних (таблиця виконана самостійно)

	1000FHD to 10000FHD	1000FHD to 1000UHD	1000UHD to 10000UHD	1000FHD to 10000UHD
MSSQLLocal	2.24	5.19	1.64	8.50
MSSQLFilestream	1.07	4.73	3.79	17.94
MSSQLFilestorage	0.89	6.36	0.95	6.04
MongoDBLocal	1.10	11.47	1.01	11.63
MongoDBFilestorage	0.87	3.80	1.49	5.68

При записі даних MSSQLFilestream показав відносно непогану стійкість до збільшення розміру файлів з FullHD до 8K UHD, але при цьому продуктивність також падала й від кількості операцій (див. рис. 4.1).

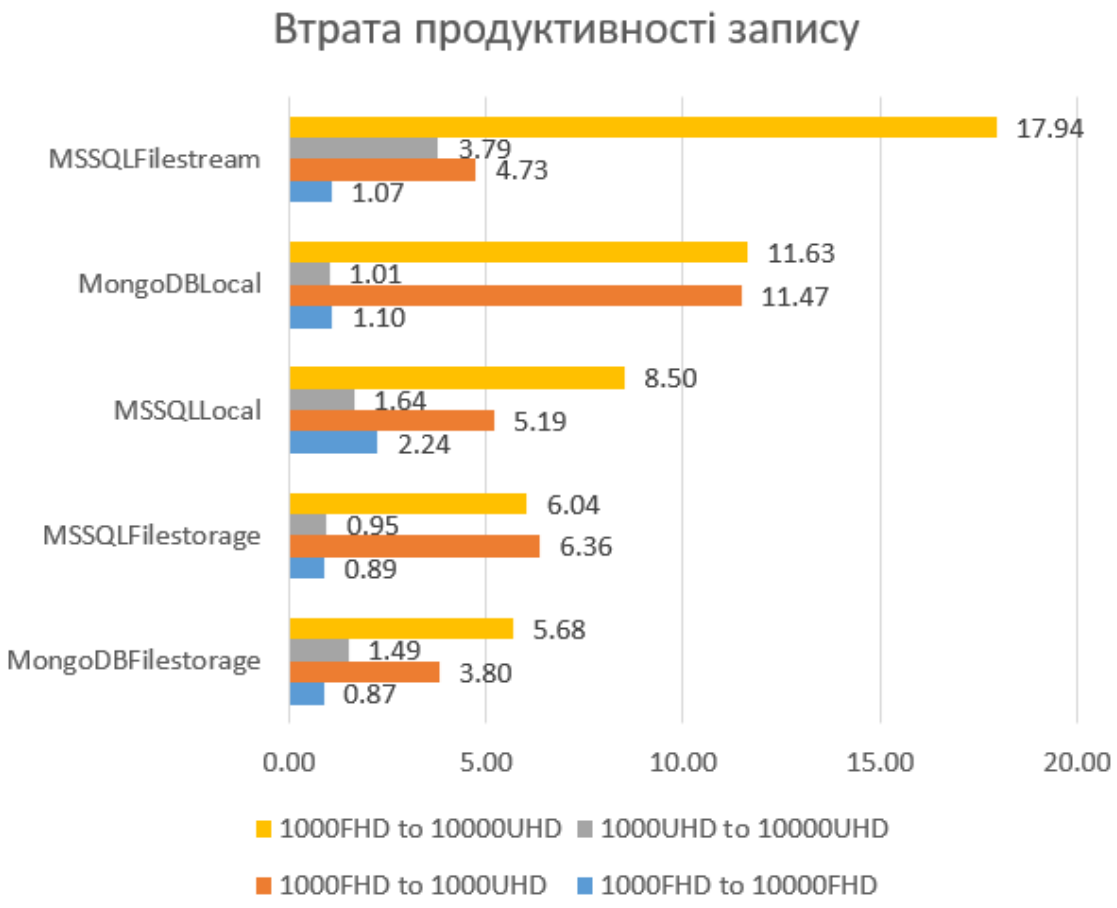

Рисунок 4.1 – Втрата продуктивності запису даних (рисунок виконаний самостійно)

Таблиця 4.5 – Втрата продуктивності читання даних (таблиця виконана самостійно)

	1000FHD to 10000FHD	1000FHD to 1000UHD	1000UHD to 10000UHD	1000FHD to 10000UHD
MSSQLLocal	1.07	500.37	1.07	534.19
MSSQLFilestream	1.23	662.22	1.23	813.33
MSSQLFilestorage	4.15	1.05	9.37	9.85
MongoDBLocal	1.65	22.02	16.38	360.73
MongoDBFilestorage	3.29	13.31	2.40	31.91

У випадку послідовного читання даних, у всіх методах окрім MSSQLFilestorage продуктивність більше знизилась через збільшення розміру файлу. Для MSSQLFilestorage розмір файлу майже не вплинув на час виконання операцій читання. При цьому основний вплив мала саме кількість записів, що свідчить про те, що при ще більшому обсязі даних MongoDBFilestorage може показати кращі результати (див. рис. 4.2).

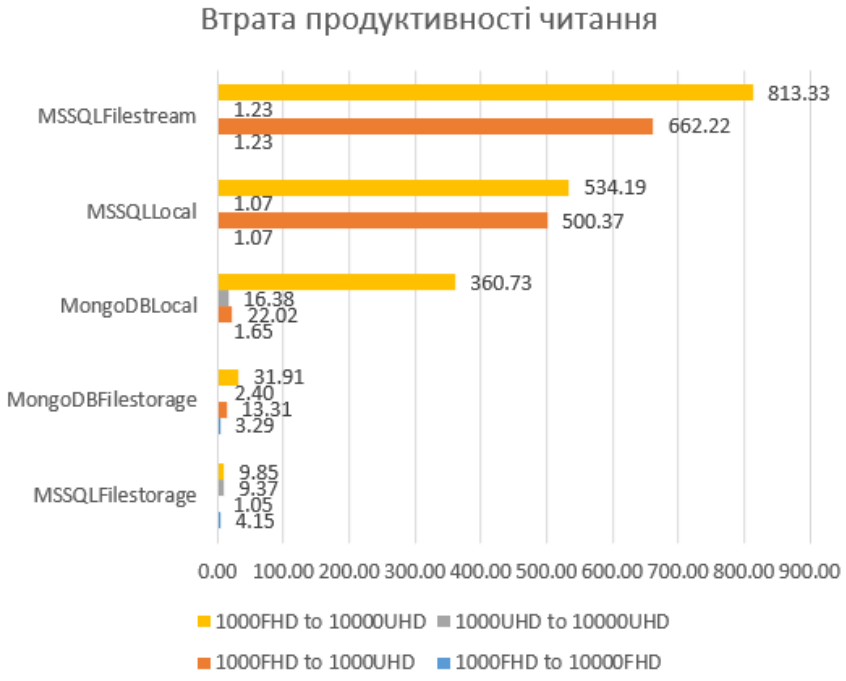

Рисунок 4.2 – Втрата продуктивності читання даних (рисунок виконаний самостійно)

Таблиця 4.6 – Втрата продуктивності фільтрації даних (таблиця виконана самостійно)

	1000FHD to 10000FHD	1000FHD to 1000UHD	1000UHD to 10000UHD	1000FHD to 10000UHD
MSSQLLocal	1.21	190.03	1.02	193.91
MSSQLFilestream	1.27	288.37	1.27	365.27
MSSQLFilestorage	2.39	8.94	1.32	11.79
MongoDBLocal	17.23	32.82	13.93	457.30
MongoDBFilestorage	5.47	2.59	3.18	8.25

Результати тестування фільтрації даних показують слабкість MongoDB до зберігання великих обсягів неструктурованих даних, де продуктивність операцій сильно знижувалась як від кількості, так і від обсягу файлів. В цьому випадку MongoDBLocal показав навіть гіршу масштабованість, ніж MSSQLFilestream (див. рис. 4.3).

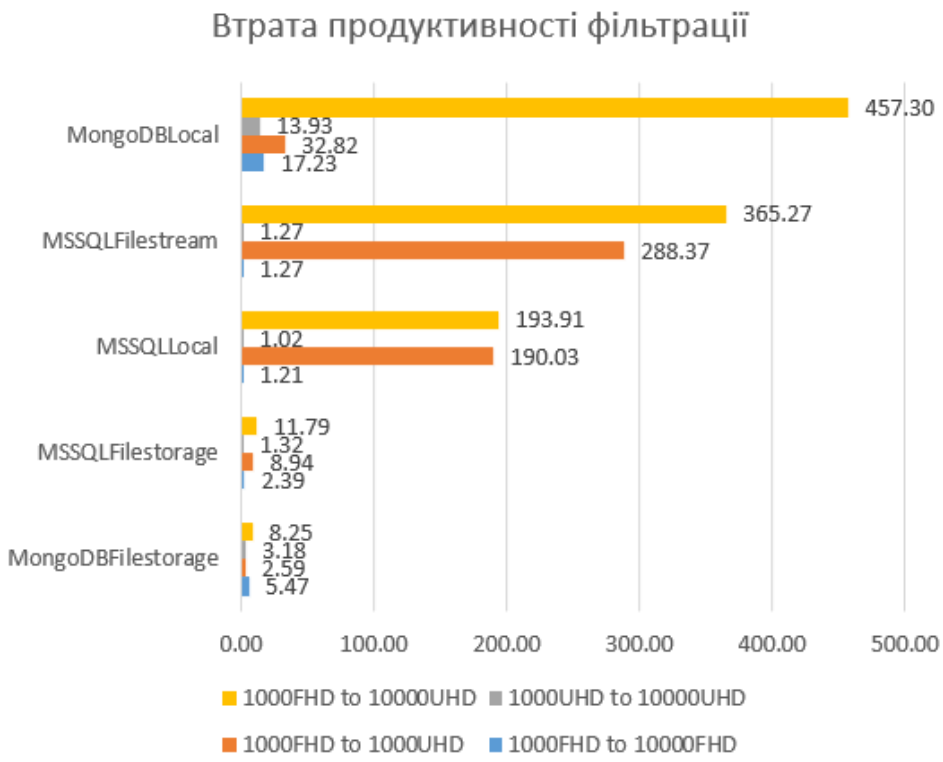

Рисунок 4.3 – Втрата продуктивності фільтрації даних (рисунок виконаний самостійно)

Отримані результати аналізу масштабованості цих методів зберігання даних свідчать про неефективність використання MS SQL Server FILESTREAM для зберігання великих обсягів неструктурованих даних через сповільнення роботи бази даних до 800 разів через накладні витрати. Цей метод вразливий як до збільшення кількості записів під час запису даних, так і до збільшення розмірів файлів.

В більшості випадків для решти методів час роботи непропорційно збільшувався через розмір файлу. Середній розмір файлу збільшився в 12.5 разів, а сповільнення в деяких випадках дійшло до 500 разів (послідовне читання при зберіганні файлів безпосередньо в базі даних MongoDB). Результати свідчать, що і реляційна СКБД Microsoft SQL Server, і нереляційна MongoDB, мають низьку масштабованість при роботі з великими обсягами неструктурованих даних.

При цьому гібридний підхід до сховища сповільнився всього в середньому в 8 разів. З урахуванням базового часу операції до 5мс та загального збільшення обсягів даних приблизно в 123 рази, найгірший результат в 41мс на операцію пошуку є прийнятним результатом.

Отже, гібридний підхід як з MS SQL Server, так і з MongoDB, виявився найкращим методом зберігання та роботи з даними великого обсягу з боку масштабованості програмної системи.

ВИСНОВКИ

В ході дослідження було проаналізовано проблемну область, існуючі методи зберігання зображень в базах даних та існуючі підходи до зберігання для реляційних та нереляційних баз даних.

Ціллю кваліфікаційної роботи було дослідження методів зберігання великого обсягу візуальних даних в базах даних. Основна мета роботи полягала в аналізі методів зберігання зображень в реляційних та нереляційних базах даних.

Методом лінійної згортки серед реляційних СУБД було обрано MS SQL Server для подальшого експериментального дослідження продуктивності зберігання великих обсягів графічної інформації та MongoDB як нереляційну базу даних.

Обрано соціальні мережі як предметну область для дослідження, зважаючи на стрімкий зріст кількості користувачів інтернету та різноманітність візуальних даних. Було запропоновано мінімальний набор сутностей (користувач – публікація – зображення), який повинен дозволити експериментально дослідити ефективність різних методів зберігання даних.

Запропоновано план експериментального дослідження методів зберігання великих обсягів візуальної інформації для реляційної та нереляційної СУБД. Для різних методів, розміру файлу та кількості файлів буде порівняно час запису файлів, час читання, час фільтрації та використання системних ресурсів.

Розроблено програмну систему на платформі ASP.NET Core, яка дозволяє експериментально виміряти продуктивність перелічених вище методів зберігання великих обсягів візуальних даних для різних конфігурацій реляційних та нереляційних баз даних.

В результаті експериментального дослідження було отримано результати, по яким можна зробити наступні висновки.

Використання Microsoft SQL Server FILESTREAM виявилось найгіршею опцією. З використанням цієї технології, було втрачено як переваги баз даних, такі як кешування актуальних даних в базах даних та швидкодію індексів, так і переваги окремого зберігання файлів через накладні витрати з боку бази даних.

Зберігання файлів безпосередньо в базах даних MS SQL Server та MongoDB показало прийнятні результати для невеликого обсягу даних, але зі збільшенням набору даних фрагментація даних та неефективність індексів значно сповільнили роботу обох СКБД, що свідчить про погану масштабованість цих методів.

Отже, гібридний метод виявився оптимальним варіантом для зберігання великих обсягів візуальних даних. Отримані як найкращі базові значення продуктивності розробленої програмної системи, так і найкращі показники масштабованості. Окреме зберігання файлів дозволило повністю використати переваги баз даних, такі як висока швидкодія, індексація.

Таким чином, проведене дослідження показало, що вибір оптимальної конфігурації гібридного сховища залежить від предметної області застосунку. Варто використовувати MS SQL Server, якщо присутні складні зв’язки між сутностями чи якщо критично важливе дотримання ACID властивостей, коли як MongoDB слід використовувати, якщо необхідні гнучкість та горизонтальна масштабованість програмної системи. Отже, доцільно провести ще додаткові дослідження гібридних підходів для різних предметних областей.

За результатами роботи опублікована стаття «Дослідження методів зберігання візуальних даних» в науково-технічному журналі «Біоніка Інтелекту», який входить в перелік фахових видань України (ВАК), категорія Б (див. Додаток В) [37].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Data never sleeps. URL: https://www.domo.com/data-never-sleeps (дата звернення: 18.04.2025).

2. Data never sleeps 12. URL: https://www.domo.com/learn/infographic/data-never-sleeps-12 (дата звернення: 18.04.2025).

3. Калашников П., Кириченко І. Гібридне сховище даних для оптимізації та покращення обробки медіафайлів. VI Всеукраїнська студентська наукова конференція «Експериментальні та теоретичні дослідження в контексті сучасної науки», 21 червня 2024 р., м. Рівне, 2024. С. 156–158. DOI: 10.62732/liga-ukr-21.06.2024.

4. Blob storage pricing. Microsoft Azure. URL: https://azure.microsoft.com/en-us/pricing/details/storage/blobs/#pricing (дата звернення: 19.04.2025).

5. Pricing. MongoDB. URL: https://www.mongodb.com/pricing (дата звернення: 19.04.2025).

6. Терещенко Г. Ю., Хіміч Є. Р. Порівняльний аналіз традиційних та гібридних моделей сховищ зображень у сфері великих даних. Біоніка інтелекту. 2022. Т. 1, № 98. С. 58–63. DOi: 10.30837/ bi.2022.1(98).07.

7. Aditya. Why SQL databases are more vertically scalable than horizontally scalable. Medium. URL: https://aditya003-ay.medium.com/why-sql-databases-are-more-vertically-scalable-than-horizontally-scalable-ef3a3f5d5f05 (дата звернення: 21.04.2025).

8. Vitess. Official Documentation. URL: https://vitess.io/ (дата звернення: 21.04.2025).

9. FILESTREAM (SQL Server). Microsoft. URL: https://learn.microsoft.com/en-us/sql/relational-databases/blob/filestream-sql-server (дата звернення: 21.04.2025).

10. SecureFiles white paper. Oracle. URL: https://www.oracle.com/a/otn/docs/securefiles-whitepaper-2009-160970.pdf (дата звернення: 21.04.2025).

11. MongoDB Replication and Sharding. GeeksforGeeks. URL: https://www.geeksforgeeks.org/mongodb-replication-and-sharding/ (дата звернення: 21.04.2025).

12. Творошенко І.С. Технології прийняття рішень в інформаційних системах: навч. посібник. – Харків: ХНУРЕ, 2021. – 120 с.

13. MySQL Documentation. MySQL. URL: https://dev.mysql.com/doc/ (дата звернення: 27.04.2025).

14. MariaDB Knowledge Base. MariaDB. URL: https://mariadb.com/kb/en/ (дата звернення: 27.04.2025).

15. SQL Server Documentation. Microsoft. URL: https://learn.microsoft.com/en-us/sql/sql-server/ (дата звернення: 27.04.2025).

16. Oracle Database Documentation. Oracle. URL: https://docs.oracle.com/en/database/ (дата звернення: 27.04.2025).

17. Db2 Database Product Documentation. IBM. URL: https://www.ibm.com/support/pages/db2-database-product-documentation (дата звернення: 27.04.2025).

18. Google Cloud Pricing Calculator. Google Cloud. URL: https://cloud.google.com/products/calculator (дата звернення: 27.04.2025).

19. Azure Database for MariaDB Pricing. Microsoft Azure. URL: https://azure.microsoft.com/en-in/pricing/details/mariadb/ (дата звернення: 27.04.2025).

20. Azure Pricing Calculator. Microsoft Azure. URL: https://azure.microsoft.com/en-us/pricing/calculator/?service=cloud-services (дата звернення: 27.04.2025).

21. Oracle Cloud Cost Estimator. Oracle. URL: https://www.oracle.com/cloud/costestimator.html (дата звернення: 27.04.2025).

22. AWS Pricing Calculator. Amazon Web Services. URL: https://calculator.aws/#/createCalculator/RDSDb2 (дата звернення: 27.04.2025).

23. GridFS. MongoDB. URL: https://www.mongodb.com/docs/manual/core/gridfs/ (дата звернення: 30.04.2025).

24. Cassandra basics. Apache Cassandra. URL: https://cassandra.apache.org/_/cassandra-basics.html (дата звернення: 30.04.2025).

25. Documentation. Neo4j. URL: https://neo4j.com/docs/ (дата звернення: 30.04.2025).

26. Azure Cosmos DB documentation. Microsoft. URL: https://learn.microsoft.com/en-us/azure/cosmos-db/ (дата звернення: 30.04.2025).

27. JDBC binary data. PostgreSQL. URL: https://www.postgresql.org/docs/7.4/jdbc-binary-data.html (дата звернення: 30.04.2025).

28. Public cloud pricing. OVHcloud. URL: https://www.ovhcloud.com/en/public-cloud/prices/ (дата звернення: 30.04.2025).

29. Pricing. Neo4j. URL: https://neo4j.com/pricing/ (дата звернення: 19.11.2024).

30. Cosmos DB capacity calculator. Microsoft Azure. URL: https://cosmos.azure.com/capacitycalculator/ (дата звернення: 30.04.2025).

31. Wikipedia statistics. Wikipedia. URL: https://en.wikipedia.org/wiki/Wikipedia:Statistics (дата звернення: 02.05.2025).

32. Create web APIs with ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-9.0 (дата звернення: 11.05.2025).

33. Overview of Entity Framework Core - EF Core. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 11.05.2025).

34. REST API Documentation Tool | Swagger UI. URL: https://swagger.io/tools/swagger-ui/ (дата звернення: 11.05.2025).

35. Home | BenchmarkDotNet: https://benchmarkdotnet.org/ (дата звернення: 17.05.2025).

36. vitalii-nikolenko/DbTest. *GitHub*. URL: https://github.com/vitalii-nikolenko/DbTest (дата звернення: 12.06.2025).

37. Кириченко І.В., Терещенко Г.Ю., Ніколенко В.В. Дослідження методів зберігання візуальних даних. Біоніка інтелекту. 2025. Т. 1, № 102. С. 20–26. DOi: 10.30837/ bi.2025.1(102).03.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

3. Калашников П., Кириченко І. Гібридне сховище даних для оптимізації та покращення обробки медіафайлів. VI Всеукраїнська студентська наукова конференція «Експериментальні та теоретичні дослідження в контексті сучасної науки», 21 червня 2024 р., м. Рівне, 2024. С. 156–158. DOI: 10.62732/liga-ukr-21.06.2024.

6. Терещенко Г. Ю., Хіміч Є. Р. Порівняльний аналіз традиційних та гібридних моделей сховищ зображень у сфері великих даних. Біоніка інтелекту. 2022. Т. 1, № 98. С. 58–63. DOi: 10.30837/ bi.2022.1(98).07.

37. Кириченко І.В., Терещенко Г.Ю., Ніколенко В.В. Дослідження методів зберігання візуальних даних. Біоніка інтелекту. 2025. Т. 1, № 102. С. 20–26. DOi: 10.30837/ bi.2025.1(102).03.