Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «Дослідження функіональних можливостей документноорієнтованих баз даних MongoDB та Google Firestore»
за освітньою програмою **12 Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи ПЗ2222: / Максим ПЛЯШКО /

Керівник: / Олександр ІВАНОВ /

Нормоконтролер: /Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент

Дніпро – 2024 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis

on the topic: «Exploring the functionality of document-oriented databases MongoDB and Google Firestore»

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ2222: /Maksym PLIASHKO/

Scientific Supervisor: /Oleksandr IVANOV/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2024

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ____КІТ
_____ Вадим ГОРЯЧКІН
_____ грудня 2023 р.

ЗАВДАННЯ

На кваліфікаційну роботу ____Магістр____
студенту Пляшку Максим Сергійовичу____.
1. Тема дипломної роботи: «Дослідження функціональних можливостей документ-орієнтованих баз даних MongoDB та Google Firestore».
Керівник роботи: Іванов Олександр Петрович
затверджені наказом 1196 ст від 05.12.2022 року
2. Строк подання студентом роботи ___.01.2024 року
3. Вихідні дані до дипломної роботи:
_____.
4. Зміст пояснювальної записки (перелік питань до розробки):
4.1. Аналітична частина: Аналіз сучасного стану дослідження за науковими літературними джерелами;
4.2. Основна частина: Часове дослідження роботи ДОБД;
5. Перелік демонстраційного матеріалу:
5.1. презентація;
5.2. демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	01.09.23 – 01.10.23	10%
2	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	01.10.22 – 10.10.22	
3	Постановка задачі, технічне завдання	01.09.23 – 15.09.23	30%
4	Розробка інструментальних засобів дослідження	15.09.23 – 01.11.23	
5	Виконання досліджень	01.10.23 – 01.11.23	60%
6	Оформлення пояснювальної записки	01.11.23 – 01.01.24	
7	Розробка демонстраційних матеріалів	10.01.24 – 15.01.24	100%
8	Подання кваліфікаційної роботи до кафедри	18.01.24	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	25.01.24	

Студент: _____ Максим ПЛЯШКО

Керівник роботи: _____ доц. Олександр ІВАНОВ

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра складається з 64., 76 рис. , 4 табл., 3 додатків, 16 джерел.

Об'єктом дослідження функціональні можливості документ-орієнтованих баз даних MongoDB та Google Firestore.

Мета роботи: дослідити роботу документ-орієнтованих баз даних MongoDB та Google Firestore. Провести дослідження областей застосування та час роботи основних функції кожної з бази даних. Формулювання рекомендацій використання доцільності використання кожної з баз даних. Проаналізувати переваги та недоліки кожної з баз даних.

Методика дослідження: теоретичний аналіз наукових і літературних джерел, тестування, замір часу виконання.

Перелік ключових слів: документ-орієнтовані бази даних, ДОБД, MongoDB, Google Firestore.

ЗМІСТ

ВСТУП

В еру інформаційних технологій великої актуальності набуває питання управлінням великими обсягами даних. Однією з сучасних ідей для вирішення цього питання виступають системи документ-орієнтованих баз даних.

Ці системи мають унікальні властивості:

- гнучкість структури даних;
- ефективна взаємодія між веб та мобільними додатками;
- легка масштабованість.

У даній роботі будуть розглянуті два провідних представники цього класу ДОБД MongoDB та Google Firestore.

Ці ДОБД мають високу продуктивність, оскільки використовують JSON-like структуру збереження даних.

Метою дослідження є аналіз функціональних можливостей документ-орієнтованих баз даних та встановлення переваг та недоліків кожної системи.

Дослідження включає аналіз функціональних можливостей кожної з обраних ДОБД, проведення навантажувального тестування , порівняння результатів виконання запитів.

Найбільша увага приділяється практичному використанню цих ДОБД, що дасть не тільки теоретичні знання про їх можливості, а також краще оцінити їх переваги та недоліки в конкретних умовах.

Таким чином, метою цього дослідження а надання комплексного огляду функціональних можливостей MongoDB та Google Firestore, визначити їхні переваги та обмеження, та забезпечити підстави для обґрунтованого вибору одного з них в залежності від вимог та конкретного контексту використання.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНОГО СТАНУ ДОСЛІДЖЕННЯ ЗА НАУКОВИМИ ЛІТЕРАТУРНИМИ ДЖЕРЕЛАМИ

1.1 Основні поняття та принципи ДОБД

Документ-орієнтовані бази даних (ДОБД) – це вид NoSQL баз даних, які забезпечують зберігання даних у вигляді певних документів, як правило у форматі JSON. Документи можуть включати в себе дані різних типів та різні структури, саме ця властивість робить їх особливо гнучкими для використання. Кожен документ може мати унікальний номер. Головна відмінність ДОБД від класичних СУБД полягає у використанні документів як одного об'єкту для маніпулюванню та зберіганню різної інформації.

Основними властивостями для ДОБД є:

- документ – об’єкт в якому зберігається інформація. Він представляє собою набір даних ключ-значення. Як правило представляються у форматі JSON;
- колекція – це група документів які зберігаються в одному місці разом. Аналогом до колекції в порівнянні з класичними СУБД є таблиці, але на відміну від таблиць колекції мають можливість зберігати дані з різною структурою;
- вкладеність та масиви – як правило документи можуть містити в собі масиви даних або навіть інші документи, що дозволяє створювати багатовимірні структури та спрощує моделювання об’єктів і відносин між ними;
- гнучкість – на відміну від класичних СУБД, ДОБД можуть працювати без жорсткої типізації даних. Це означає що документи можуть мати різний набір полів з різними типами, необов’язкові поля які додаватися уже до існуючих документів;
- ідентифікатор документа – кожен документ має унікальний номер який дозволяє швидко знаходити його в межах колекції, та працювати з ним;

- читання та запис – на відміну від класичних СУБД, ДОБД використовують асинхронний підхід для зчитування, запису та модифікації даних;
- спрощені запити – запити в ДОБД є інтуїтивно зрозумілими, що полегшує роботу з ДОБД для розробників;
- великий обсяг даних – ДОБД розроблені саме для роботи з великими обсягами інформації, що робить їх ефективними для застосунків де використовуються дані які постійно змінюються [1].

1.2 Функціональні можливості MongoDB

MongoDB – це одна з широко використовуваних ДОБД, яка має високу продуктивність та високу швидкодію опрацювання великих за обсягами неструктурованими даними.

MongoDB базується на концепції документів, які відповідають об’єктам формату BSON (Binary JSON). Ці документи групуються у колекції що на відміну від табличної структури дозволяю використовувати та зберігати дані більш універсально та гнучко. MongoDB також використовує мову запитів яка дозволяє виконувати різні операції для видалення, додавання модифікування, читання даних. Запити виконуються за допомогою внутрішньої мови яка має зрозумілий контекст MongoDB Query Language (MQL) [2].

Для того щоб додати нові документи достатньо написати відповідний запит.

Структура запиту має вигляд:

```
[
    { <документ 1> },
    { <документ 2> },
    … ,
    { <документ N> }
]
```

Приклад з запитом додавання 3х документів представлений зображено на рисунку 1.1:

```
[
        { "Key": "Element1" },
        { "Key": "Element2" },
        { "Key": "Element3" }
]
```

Рис. 1.1 — Приклад запиту додавання документів до БД

Результат роботи запиту додавання документів до колекції наведено на рисунку 1.2:

	_id ObjectId	Key String
1	ObjectId('658ac3ef1a34fff…	"Element1"
2	ObjectId('658ac3ef1a34fff…	"Element2"
3	ObjectId('658ac3ef1a34fff…	"Element3"

Рисунок 1.2 — Результат роботи запиту додавання документів до БД

Для виконання запитів фільтрації можна використовувати вікно фільтрації. Загальний вигляд вікна фільтрації наведено на рисунку 1.3:

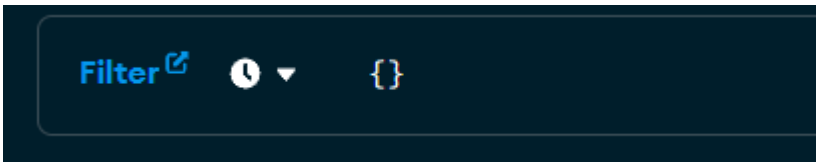


Рисунок 1.3 — Загальний вигляд вікна фільтрації

Для роботи з складнішими запитами такі як: оновлення видалення документів потрібно використовувати _MONGOSH. Загальний вигляд консолі _ MONGOSH наведено на рисунку 1.4:

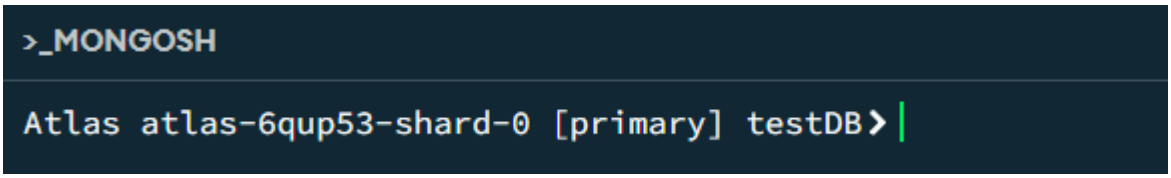


Рисунок 1.4 — Загальний вигляд консолі _ MONGOSH

_ MONGOSH представляє собою звичайний термінал для роботи з запитами. Загальний вигляд підказок в консолі _ MONGOSH наведено на рисунку 1.5:

Рисунок 1.5 — Загальний вигляд підказок в консолі _ MONGOSH

Для того щоб отримати результат роботи фільтру треба в вікно фільтру написати запит:

{ <умова> }

Запит з отриманням документів де ключ Key дорівнює Element1 наведено на рисунку 1.6:

Рисунок 1.6 — Приклад запиту фільтрації документів з БД

Результат роботи запиту отримання документів використовуючи фільтр наведено на рисунку 1.7:

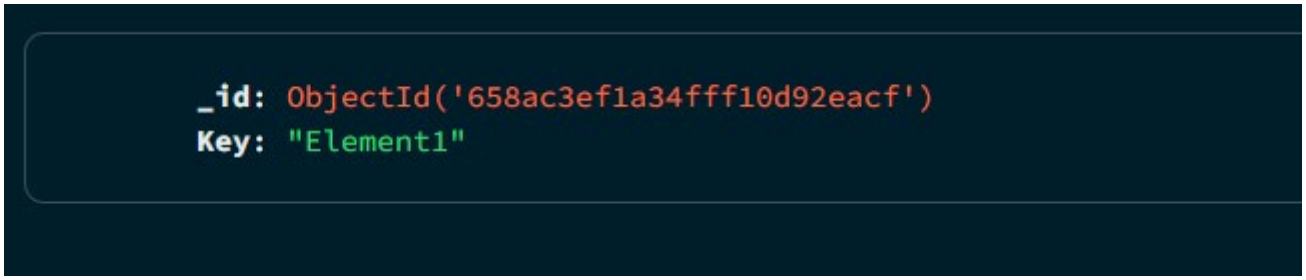


Рисунок 1.7 — Результат роботи запиту отримання документів з БД

Для того щоб отримати результат роботи складного фільтру OR треба в вікно фільтру написати запит [3]:

```
{ $or [
        {<умова1>},
        {<умова2>},
        {<умова3>}
] }
```

Запит з отриманням документів де ключ Key дорівнює Element1 АБО Key дорівнює Element2 зображено на рисунку 1.8:

Рисунок 1.8 — Приклад запиту отримання документів з умовою АБО з БД

Результат роботи запиту отримання документів використовуючи складний фільтр АБО наведено на рисунку 1.9:

Рисунок 1.9 — Результат роботи запиту отримання документів з умовою АБО з БД

Для того щоб виконати операцію оновлення документів потрібно в _MONGOSH викликати функцію updateMany. Функція має таку сигнатуру[4]:

updateMany(<фільтр/складний фільтр>, <нові значення>)

Запит з на додавання нового поля NewKey з значенням «55» до документів де ключ Key дорівнює Element1 АБО Key дорівнює Element2 зображено на рисунку 1.10:

```
> db.getCollection("123").updateMany( {
        $or: [
           { Key: "Element2" },
           { Key: "Element1" }
        ]
    },
    {
        $set: { NewKey: "55" }
    }
 )
```

Рисунок 1.10 — Приклад запиту оновлення документів з умовою АБО в БД

Під час виконання запиту в _ MONGOSH сервер повернув відповідь в вигляді структури даних. Загальний вигляд відповіді сервера на функцію updateMany зображено на рисунку 1.11:

```
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0
  }
```

Рисунок 1.11 — Відповідь сервера на запит оновлення

Відповідь складається з полів:

– acknowledged: true: Це поле підтверджує, що запит був виконаний і отриманий сервером MongoDB. Якщо значення - true, це означає, що операція була успішно виконана.
– insertedId: null: Значення null у цьому полі вказує на те, що не було вставлено нового документа, або, в інших операціях (наприклад, оновлення), не було створено нового запису.

- matchedCount: 2: Це кількість документів, які відповідають умовам пошуку (у разі операцій оновлення чи видалення). Значення matchedCount рівне 2 вказує що лише 2 документа відповідає умовам пошуку.
- modifiedCount: 2: Кількість документів, які були змінені під час операції оновлення. Значення modifiedCount рівне 2, вказує на те, що було змінено лише 2 документів.
- upsertedCount: 0: Кількість нових документів, які були вставлені (у випадку оновлення з параметром "upsert"). Значення upsertedCount рівне 0, означає відсутність нових вставлених документів.

Результат роботи запиту оновлення документів використовуючи фільтр наведено на рисунку 1.12:

Рисунок 1.12 — Результат роботи запиту оновлення документів з умовою АБО з БД

Для того щоб отримати результат роботи складного фільтру AND треба в вікно фільтру написати запит [5]:

{ $and [
{<умова1>},
{<умова2>},
{<умова3>}
] }

Запит з отриманням документів де ключ Key дорівнює Element1 І NewKey дорівнює 55 зображено на рисунку 1.13:

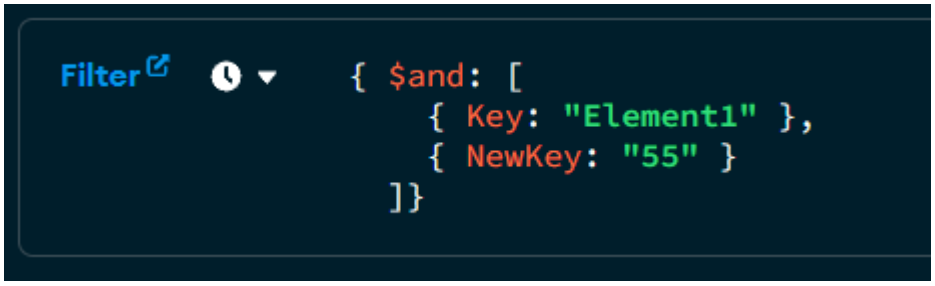


Рисунок 1.13 — Приклад запиту отримання оновлення документів з умовою І в БД

Результат роботи запиту отримання документів використовуючи складний фільтр І наведено на рисунку 1.14:

Рисунок 1.14 — Результат роботи запиту отримання документів з умовою І з БД

Для того щоб виконати запит видалення документів потрібно в _ MONGOSH викликати функцію deleteMany. Функція має таку сигнатуру [6]:

deleteMany (<фільтр/складний фільтр>)

Запит на видалення документів де ключ Key дорівнює Element1 І NewKey дорівнює 55 зображено на рисунку 1.15:

```
> db.getCollection("123").deleteMany( {
        $and: [
            { Key: "Element2" },
            { NewKey: "55" }
        ]
    }
)
```

Рисунок 1.15 — Приклад запиту отримання видалення документів з умовою I в БД

Під час виконання запиту в _ MONGOSH сервер повернув відповідь в вигляді структури даних. Загальний вигляд відповіді сервера на функцію deleteMany зображено на рисунку 1.16:

```
< {
    acknowledged: true,
    deletedCount: 1
  }
```

Рисунок 1.16 — Відповідь сервера на запит видалення

Відповідь складається з полів:

– acknowledged: true: Це поле підтверджує, що запит був виконаний і отриманий сервером MongoDB.
– deletedCount: 1: Значення 1 у цьому полі вказує на те, було видалено лише один документ.

Результат роботи запиту видалення документів з БД використовуючи складний фільтр I наведено на рисунку 1.17:

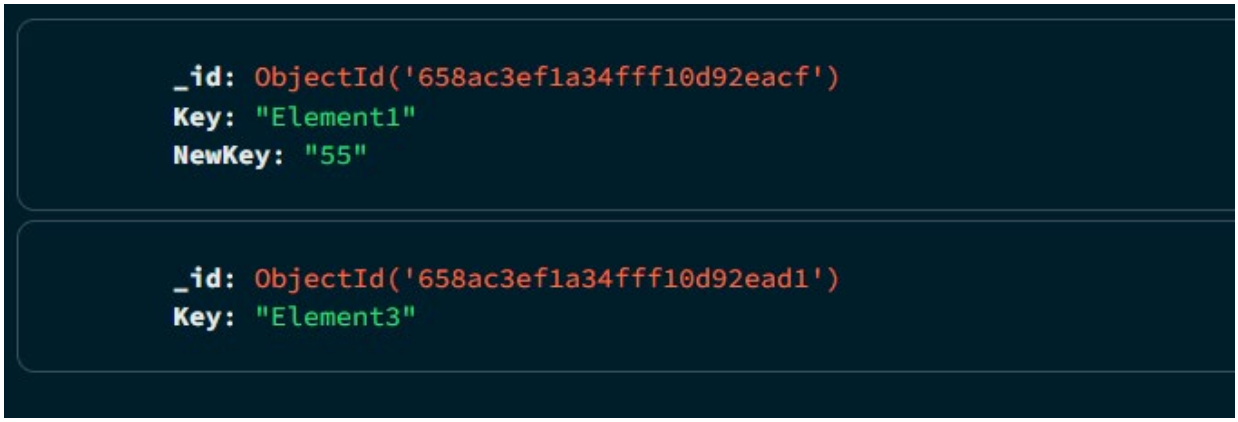


Рисунок 1.17 — Результат роботи запиту видалення документів з БД

Для того щоб видалити всі документи з колекції потрібно викликати функцію deleteMany з пустим фільтром

Запит на видалення всіх документів зображено на рисунку 1.18:

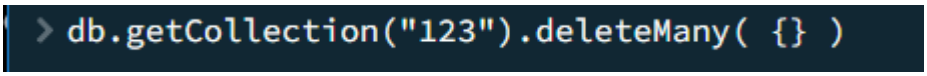


Рис. 1.18 — Приклад запиту видалення всіх документів в БД

Під час виконання запиту в на видалення двох останіх документів з БД використовуючи термінал_MONGOSH сервер повернув відповідь на результат роботи функції. Загальний вигляд відповіді сервера на функцію deleteMany зображено на рисунку 1.19:

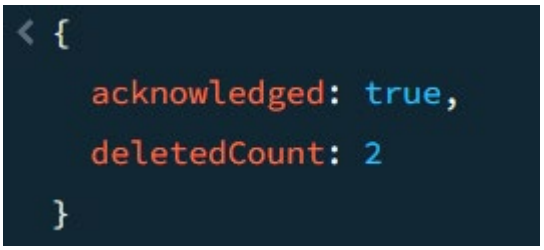


Рис. 1.19 — Відповідь сервера на запит видалення

Результат роботи запиту видалення всіх документів з БД наведено на рисунку 1.20:

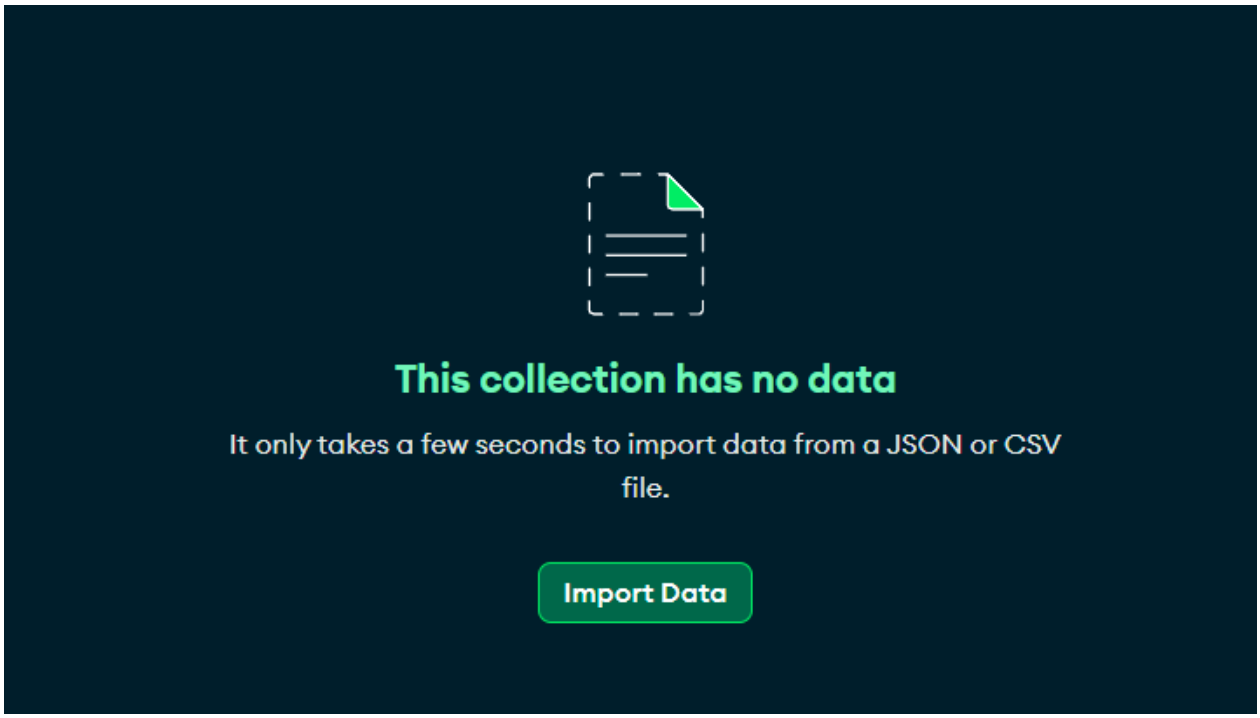


Рис. 1.20 — Результат роботи запиту отримання документів з умовою або з БД

MongoDB дозволяє отримувати певні поля документів без потреби зчитування всього документу — застосування методу індексації для покрашення швидкодії виконання запитів, що робить пошук та сортування ефективним при великому обсягу даних. Також дозволяється створювати документи без заданої структури, що дозволяє зберігати дані різних типів та структур в межах однієї колекції. Також є можливість використання транзакцій, що дозволяє використовувати складні запити для обробки даних, що дозволяє виконувати групу операцій бази даних як єдину атомарну одиницю. Всі операції в транзакції виконуються атомарно, тобто або всі вони успішно завершуються, або жодна з них не виконується. Це гарантує, що база даних завжди залишається в консистентному стані. Транзакції забезпечують консистентність даних, забезпечуючи виконання всіх правил та обмежень бази даних. Транзакції ізолюють одну від одної, щоб запобігти взаємному впливу, і забезпечити, що результати однієї транзакції не стають видимими для інших транзакцій, поки транзакції не завершать виконання. Після успішного завершення транзакції її результати зберігаються і залишаються стійкими до збоїв або перезавантаження системи.

MongoDB широко використовується у різних галузях таких як, веб додатки, мобільна розробка, аналітика, фінанси та інші.

1.3 Функціональні можливості Google Firestore

Google Firestore – одна з передових ДОБД, яка розроблена для роботи з хмарними середовищами для швидкого та масштабованого доступу до інформації. Google Firestore зберігає дані у вигляді колекцій які містять в собі документи — набір полів та значень, які зберігаються у форматі JSON.

Колекції відповідають групам документів які можна створювати динамічно. Google Firestore як і MongoDB використовує індексацію документів. Оскільки в Google Firestore можна створювати індекси для кожного документу в колекції, це сприяє полегшенню виконанню запитів. Одна з ключових можливостей Google Firestore – це можливість відстежування зміни в реальному часі, це дозволяє отримувати сповіщення про зміну даних без потреби постійного опитування[7].

Також однією з можливостей Google Firestore є інтеграція з сервісами Google Cloud Platform, що надає потужність використовувати інші сервіси, такі як:

- Google Compute Engine (GCE) – надає віртуальні машини різних конфігурацій та операційних систем для обчислення в хмарному середовищі;
- Google App Engine (GAE) – платформа для розгортання та масштабування веб-додатків автоматично, без ведення інфраструктури;
- Google Kubernetes Engine (GKE) – управління контейнерами з використанням системи керування контейнерами Kubernetes;
- Google Cloud Storage – служба зберігання об'єктів для зберігання та вибору даних в хмарному середовищі;
- Google Cloud Bigtable – розподілена служба бази даних NoSQL, призначена для великого обсягу структурованих даних;
- Google Cloud SQL – управління реляційною базою даних MySQL та PostgreSQL в хмарному середовищі;

- Google Cloud Datastore – розподілена служба бази даних NoSQL для зберігання та запитів даних;
- Google Cloud Pub/Sub – служба обміну повідомленнями для побудови розподіленого та масштабованого хмарного архітектурного рішення;
- Google Cloud BigQuery – служба аналітики для швидкого та масштабованого аналізу великих обсягів даних за допомогою SQL-запитів;
- Google Cloud Machine Learning Engine – сервіс для створення, тренування та розгортання моделей машинного навчання на хмарних ресурсах;
- Google Cloud Vision API, Google Cloud Natural Language API – сервіси для обробки зображень та тексту за допомогою штучного інтелекту;
- Google Cloud IoT Core – служба для збору, аналізу та управління даними в Інтернеті речей (IoT).

Одним з основних питань також виступає захист даних в БД. Google Firestore надає інструменти для надійного захисту даних та обмеження доступу до даних. Google Firestore дозволяє налаштувати рівень доступу для кожного документа чи колекції в БД. Firestore підтримує можливість роботи в офлайн режимі, що є відміним вибором для додатків, яким потрібен постійний доступ до даних навіть без інтернет з’єднання.

Google Firestore є найоптимальнішим рішенням для роботи з БД в сукупності з Google Cloud Platform.

1.4 Порівняльний аналіз між MongoDB та Google Firestore

Порівняння функціональних можливостей між MongoDB та Google Firestore дозволяє визначити переваги та недоліки кожної ДОБД.

Основні аспекти порівняння:

- модель даних:
  - MongoDB використовує документ-орієнтовану модель даних з можливостями додавати вкладені об'єкти та масиви у документах. Дозволяє обробляти неструктуровані;

– Google Firestore також використовує документ-орієнтовану модель, де дані організовані у вигляді колекцій та документів. Також підтримує вкладеність та масиви;

– індексація та швидкість:

– MongoDB дозволяє створювати індекси для полів у колекції документів, що сприяє зростанню швидкодії виконанню запитів. Потребує уважності для оптимізації для індексів, для забезпечення максимальної продуктивності для роботи з середнім та великим обсягом даних;

– Google Firestore створює індекси автоматично для всіх полів документу у колекції, що сприяє ефективності роботи при управлінням невеликим та середнім обсягом даних;

– масштабування:

– MongoDB підтримує можливість як вертикальне та горизонтальне масштабування даних, що надає переваги в гнучкості у виборі стратегії масштабування;

– в свою чергу Google Firestore орієнтоване на горизонтальне масштабування, яке спрямоване на максимальну ефективність при роботі з сервісами Google Cloud Platform;

– реплікація та Надійність:

– MongoDB має розвинуті механізми реплікації та забезпечує високий рівень надійності даних;

– як і MongoDB, Google Firestore використовує реплікацію та надає високий рівень надійності;

– транзакції:

– MongoDB підтримує основні операції для роботи з БД. Групує всі операції в атомарні блоки;

– недоліком Google Firestore в поточному порівнянні є відсутність логічної операції OR, що в певних випадках сприяє ускладненню розробки

додатків з використанням цього сервісу. Також підтримує атомарність операцій;

– специфічні особливості:

– MongoDB широко використовується у сферах веб-розробки, фінансів та аналітики. Надає гнучку схему даних та велику кількість опцій для оптимізації роботи з БД;

– Google Firestore забезпечує відстеження змін у реальному часі, що робить його ефективним для реактивних додатків де потрібно швидко дізнатися про нові дані. Має великий рівень інтеграції з Google Cloud Platform.

1.5 Обмеження MongoDB та Google Firestore

Кожна із представлених ДОБД має певний ряд обмежень.

Обмеження MongoDB:

– розмір документу - MongoDB дозволяє зберігати документи розміром не більше 16 мегабайт. Це включає вкладені об’єкти в межах одного документу. Для роботи з великими документами потрібно мати обдуману та оптимальну стратегію або розбити на більш малі документи;
– індексація - MongoDB дозволяє створювати свої індекси для більш оптимальної роботи з запитами однак завелика кількість неоптимізованих індексів може призвести до значного збільшення розміру бази даних та вплинути на швидкодію при роботі з даними;
– одночасні з'єднання - MongoDB дозволяє одночасно створювати декілька з’єднань, але кількість одночасних з’єднань з БД залежить від її конфігурації та версії. На поточний час безкоштовна конфігурація обмежена опрацюванням не більше ніж 500 з’єднань одночасно;
– розмір бази даних - MongoDB має внутрішнє сховище розмір якого залежить від конфігурації. На поточний час безкоштовна конфігурація обмежена розміром сховища не більше ніж 512 МБ.

- Обмеження Google Firestore:
- розмір бази даних - Google Firestore також як MongoDB і має внутрішнє сховище розмір якого залежить від конфігурації. На поточний час з безкоштовна конфігурація обмежена розміром сховища не більше ніж 1024 МБ що вдвічі більше ніж у MongoDB.
- одночасні операції - хоча Firestore не обмежує кількість одночасних з’єднань, сервіс має обмеження через операцій читання та запису на день. На поточний час з безкоштовна конфігурація має обмеження:
    - на створення та модифікування документу - 20000 документів на добу
    - на видалення документу – 20000 документів на добу
    - на читання документів – 50000 документів на добу
- обмеження на індексацію - На відміну від MongoDB, Firestore обмежує кількість використання індексів. Через створення та використання індексів може бути стягнута додаткова оплата, тому кількість індексів слід розглядати з урахуванням ефективності, потреби та вартості.
- зміни в реальному часі - Firestore забезпечує інформування про зміни в реальному часі, але при занадто великій кількості змін може виникнути затримка в їхньому отриманні. Це важливо враховувати при розробці додатків, які використовують реальний час для оновлень.

Висновки до розділу 1

Перший розділ містить у собі основні властивості документ-орієнтованих баз даних, опис кожної з обраних документ-орієнтованих баз даних, порівняльний аналіз та функціональні обмеження кожної з бази даних.

Хмарні документ-орієнтовані бази даних виступають потужним інструментом в розробці різних за рівнем складності додатків. Огляд документ-орієнтованих баз даних дозволив визначити яка з баз даних краще підходить під вирішення певної проблеми.

Однак, для досягнення максимальної ефективності потрібно розуміти не лише функціональні можливості кожної з баз даних а також розуміти обмеження кожної з баз даних. На приклад якщо достатньо лише зберігати та читати інформацію з БД краще використовувати MongoDB, як що виникає потреба постійно модифікувати інформацію та відстежувати зміни в інформації, краще використовувати Google Firestore.

РОЗДІЛ 2 ОБГРУНТУВАННЯ МЕТОДІВ ДОСЛІДЖЕННЯ

Для дослідження проблемної області буде створено віконний додаток.

2.1 Основні функціональні вимоги

- Зчитування та запис даних: дослідження повинно охоплювати здатність обох баз даних до зчитування та запису даних, включаючи операції вставки, оновлення та видалення записів;
- вкладеність даних: потрібно дослідити здатність баз даних обробляти вкладені дані та робити запити до вкладених полів;
- транзакції та атомарність: дослідження має охоплювати підтримку транзакцій та гарантування атомарності операцій
- захист даних: має бути досліджена система захисту даних, включаючи доступ до даних та автентифікацію користувачів.

Ці вимоги до функціональних характеристик дослідження документ-орієнтованих баз даних MongoDB та Google Firestore допоможуть зрозуміти їхні можливості та вибрати найкращу платформу для конкретних потреб проекту.

2.2 Вхідні данні

Вхідні данні до віконного додатку:

- дані про структуру документів: опис структури документів, які будуть зберігатися в базах даних, включаючи назви полів, типи даних та вкладені структури;
- обсяг даних: кількість та об'єм даних, які планується зберігати в базах даних, включаючи кількість записів та обсяг текстової і бінарної інформації;
- методи доступу до даних: опис методів доступу до даних, включаючи створення, зчитування, оновлення та видалення документів;

- сценарії використання: визначення конкретних сценаріїв використання, таких як додавання нових даних, пошук за певними умовами, оновлення та видалення записів;
- серверні налаштування: інформація про конфігурацію серверів, на яких розгорнуті бази даних, включаючи обсяг доступної пам'яті, кількість ядер процесора та інші параметри.

2.3 Вихідні дані

Вихідні данні до віконного додатку:

- результати дослідження функціональності: звіт або документ, який містить результати дослідження функціональних можливостей обох баз даних, включаючи порівняльний аналіз їхніх можливостей;
- рекомендації для вибору бази даних: інформація про те, яка з баз даних, MongoDB або Google Firestore, найкраще відповідає вимогам та потребам проекту, з вказівкою на переваги та недоліки кожної з них;
- 16: надається план впровадження ДОБД, включаючи методи міграції даних та інші необхідні кроки.

Висновки до розділу 2

Другий розділ містить у собі основні функціональні вимоги до додатку, вхідні та вихідні дані додатку

Основні вимоги до додатку включають у себе зчитування та запис інформації до кожної з баз даних, дослідження здатності баз даних працювати зі вкладеною інформацією та як це впливає на ефективність, дослідження роботи транзакцій в базах даних, та можливість захисту інформації в базах даних.

Основні вимоги до вхідних даних включають у себе опис структури документу який буде збережений до баз даних, об’єм інформації який буде зберігатися в базі даних, реалізація основних операцій роботи над базами даних що включають в себе створення, видалення, отримання та оновлення інформації

з баз даних, опис серверних налаштувань для коректної роботи додатку з хмарними середовищами а також сценарій використання додатку.

Основні вимоги до вихідних даних складаються з дослідження функціональних можливостей документ-орієнтованих баз даних, порівняльний аналіз можливостей кожної з баз даних на конкретному прикладі, рекомендації щодо вибору бази даних базуючись на отримані результати під час дослідження, а також план впровадження документ-орієнтованих баз даних.

РОЗДІЛ 3 РОЗРОБКА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ОЦІНКИ РОБОТИ БАЗ ДАНИХ

Метою даного дослідження виступає аналіз функціональних можливостей документ-орієнтованих баз даних MongoDB та Google Firestore. Проектування інструментальних засобів буде виконуватись за допомогою UML-діаграм. Unified Modeling Language, або UML — уніфікована мова візуального моделювання систем, у т.ч. програмного забезпечення. UML можливо використовувати для візуального моделювання за будь-яких обраних методології проектування.

3.1 Вибір мови програмування та середовища розробки

За допомогою технології .NET можна просто створювати віконні додатки. Також платформа .NET надає можливість легко використовувати функціонал бібліотек для роботи з MongoDB та Google Firestore. В майбутньому для точного аналізу даних буде використовуватися бібліотека NLog яка надає змогу гнучкого та легкого налаштування логування потрібної інформації в вихідний.

На сьогоднішній момент мова програмування C# одна з найпотужніших мов, що швидко розвиваються і затребувана в IT-галузі. Зараз на ній пишуться різні програми: від невеликих десктопних програм до великих веб-порталів і веб-сервісів, що обслуговують щодня мільйони користувачів.

C# є об'єктно-орієнтованою мовою програмування що надає можливість легко та зрозуміло використовувати функціонали класів не аналізуючи функціонал бібліотек. C# підтримує поліморфізм, наслідування, навантаження операторів та статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішити завдання з побудови великих, але в той же час гнучких, масштабованих і додатків, що розширюються[8].

3.2 Формалізація задачі

Формалізація задачі була представлена у вигляді діаграми діяльності на рисунку 3.1:

Рисунок 3.1 — діаграма діяльності аналізу функціональних можливостей баз даних

3.3 Особливості мови C# для використання у розробці

C# займає одне з перших місць як найпопулярніша мова програмування завдяки своїм багатофункціональним можливостям та гнучкості. C# дозволяє розробникам створювати різноманітні програми, від веб-додатків на платформі ASP.NET та ASP.NET Core до десктопних застосунків за допомогою Windows Forms та WPF, а також мобільних додатків через Xamarin. Це робить C# відмінним вибором для розробників, які працюють в різних галузях. Стійка інтеграція з іншими продуктами та технологіями Microsoft, такими як Visual Studio, Azure Cloud, і .NET Framework, надає розробникам широкий спектр інструментів для розробки, тестування та впровадження додатків. Також C# використовується у хмарній розробці завдяки своїй використанню на платформі Azure. Розробники можуть використовувати різні сервіси Azure для створення ефективних та масштабованих хмарних рішень. C# може бути використана для створення крос-платформових мобільних додатків для iOS, Android та Windows. Розробники можуть використовувати спільний код для зручної розробки та підтримки на різних платформах. C# підтримує концепції об'єктно-орієнтованого програмування, що полегшує організацію коду та сприяє його повторному використанню. Підтримка автоматичного управління пам'яттю та вбудовані механізми безпеки забезпечують стійкість та ефективність програм. Існує велика та активна спільнота розробників C#, яка спільно працює над розвитком мови та підтримкою інших розробників. Це включає в себе форуми, блоги, інструкції та різні ресурси, які полегшують навчання та розвиток.

За таких умов C# залишається однією з передових мов програмування завдяки своїм різноманітним функціональним можливостям та широкому застосуванню[9].

3.4 Особливості роботи з MongoDB

Для початку роботи з MongoDB потрібно створити обліковий запис на офіційному сайті MongoDB та створити базу даних. Після створення облікового

запису та створення бази даних потрібно зберегти посилання до БД яке буде створюватися з’єднання.

Для використання функціоналу MongoDB був створений клас контролер mongoDBController. В приватні поля класу було додано поля назви бази даних, назви колекції до якої цей модуль буде звертатися та клієнт в якому зберігається системна інформація стосовно з’єднання з БД, в який потрібно передати отримане посилання[10]. Конфігуруючі поля класу наведено на рисунку 3.2:

```
private readonly string databaseName = "testDB";
private readonly string collectionName = "123";
Ссылок: 7
private MongoClient _client => new MongoClient("mongodb+srv://max467364:maxsys2001@cluster0.zjpwsib.mongodb.net/");
```

Рисунок 3.2 — Конфігуруючі поля класу

Для того щоб додати документ до колекції треба використати функції InsertOne, яка приймає як аргумент документ який ми хочемо додати до колекції, документ повинен мати тип BsonDocument [11]. Функція додавання документу до БД представлена на рисунку 3.3:

```
_collection.InsertOne(variationDocument);
```

Рисунок 3.3 — Внутрішня функція додавання документу до БД

Для того щоб отримати документи з БД потрібно використати функцію Find, яка приймає в якості параметра фільтр через який ми будемо шукати потрібні нам документи[12]. Якщо вказати порожній фільтр то буде повернута всі документи в колекції. Функція отримання документу з БД представлена на рисунку 3.4:

```
var documents = _collection.Find(filter).Project(projection).ToList();
```

Рисунок 3.4 — Внутрішня функція отримання документу з БД

Для видалення документів з БД потрібно використати функцію DeleteMany яка в якості аргументів приймає фільтр який вказує на документи які задовольняють умовам фільтра. Якщо вказати порожній фільтр то видаляться всі

документи в колекції. Функція видалення документу до БД представлена на рисунку 3.5:

```
_collection.DeleteMany(filter);
```

Рисунок 3.5 — Внутрішня функція видалення документу з БД

Для оновлення даних в колекції потрібно використати функцію UpdateMany, яка приймає в якості аргументів фільтр який вказує на документи які задовольняють умові, та нові дані які будуть записані в відповідні документи в колекції. Функція оновлення документу в БД представлена на рисунку 3.6:

```
_collection.UpdateMany(filter, update);
```

Рисунок 3.6 — Внутрішня функція оновлення документу з БД

3.5 Особливості роботи з Google Firestore

Для початку роботи з Google Firestore потрібно створити обліковий запис на офіційному сайті FireBase та створити базу даних. Після створення облікового запису та створення бази даних потрібно створити та зберегти конфігураційний файл в якому зберігається потрібна інформація для роботи з БД [13]. Загальний вигляд конфігуруючого файлу наведено на рисунку 3.7:

Рисунок 3.7 — Загальний вигляд конфігураційний файлу

Також важливим моментом є додання файлу конфігурації до вихідної збірки додатку. Дії конфігуруючого файлу представленні на рисунку 3.8:

Действие при сборке	Содержимое
Имя файла	firestore.json
Копировать в выходной каталог	Всегда копировать

Рисунок 3.8 — Налаштування дій файлу для збірки

Для використання функціоналу Google Firestore був створений клас контролер firestoreController. В приватні поля класу було додано поля назви проекту, назви колекції з якою буде працювати модуль та поле шляху до конфігураційного файлу відносно .exe файлу додатка [13]. Конфігуруючі поля класу наведено на рисунку 3.9:

```
string projectName = "test-1de66";
string keyFilePath = Path.Combine("./firestore.json");
string collection = "qwe";
```

Рисунок 3.9 — Конфігуруючі поля класу

Для того щоб додати документ до колекції треба використати функції SetAsync, яка приймає як аргумент документ який ми хочемо додати до колекції, документ повинен мати тип Dictionary<string, object> [14]. Також для очікування кінця роботи функції було додано функцію Wait, яка очікує припинення роботи функції. Функція додавання документу до БД представлена на рисунку 3.10:

```
docRef.SetAsync(data).Wait();
```

Рисунок 3.10 — Внутрішня функція додавання документу до БД

Для того щоб отримати документи з БД потрібно використати функцію GetSnapshotAsync. Отримані результати потрібно зберігати в форматі QuerySnapshot[15]. Функція отримання документів з БД представлена на рисунку 3.11:

```
var snapshot = await condition.Value.GetSnapshotAsync().ConfigureAwait(false);
```

Рисунок 3.11 — Внутрішня функція отримання документу з БД

Для видалення документів з БД потрібно спочатку отримати снапшот документів які задовольняють умові видалення потім виконати функцію DeleteAsync для кожного з отриманих документів[16]. Функція видалення документу до БД представлена на рисунку 3.12:

```
QuerySnapshot querySnapshot = await query.GetSnapshotAsync().ConfigureAwait(false);

foreach (DocumentSnapshot documentSnapshot in querySnapshot.Documents)
{
    DocumentReference documentReference = documentSnapshot.Reference;
    documentReference.DeleteAsync().Wait();
}
```

Рисунок 3.12 — Внутрішня функція видалення документу з БД

Для оновлення даних в БД документів з БД потрібно спочатку отримати снапшот документів які задовольняють умові видалення потім виконати функцію Update, яка приймає в якості аргументів дані які потрібно додати до файлу. Функція оновлення документу в БД представлена на рисунку 3.13:

```
foreach (DocumentSnapshot documentSnapshot in querySnapshot.Documents)
{
    DocumentReference documentReference = documentSnapshot.Reference;
    await documentReference.UpdateAsync(newData);

}
```

Рисунок 3.13 — Внутрішня функція оновлення документу з БД

3.6 Тестування програми

Перевіримо можливість додатку працювати з базами даних. Додавання документів до бази даних Додаймо до баз даних 10 документів які маю таку структуру:

Key1: value1

Key2: value1 або value2

Вікно створення структури документу наведено на рисунку 3.14:

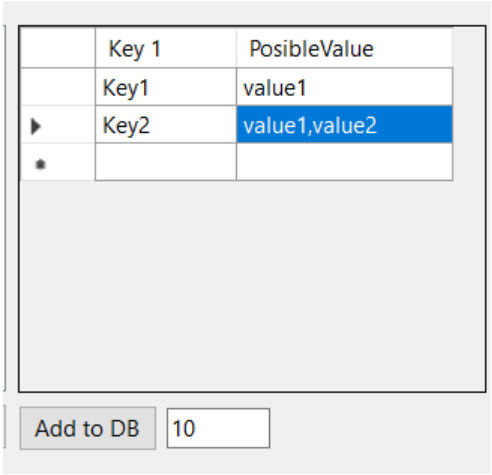


Рисунок 3.14 — Вікно створення структури документу

В БД mongoDB додалося 10 документів з відповідними полями, що вказує на те що функція додавання коректно працює з mongoDB. Результат роботи операції додавання в хмарну базу даних MongoDB наведена на рисунку 3.15:

123

	_id ObjectId	Key1 String	Key2 String
1	ObjectId('6595eec7003dea2…	"value1"	"value1"
2	ObjectId('6595eec7003dea2…	"value1"	"value1"
3	ObjectId('6595eec7003dea2…	"value1"	"value1"
4	ObjectId('6595eec7003dea2…	"value1"	"value2"
5	ObjectId('6595eec7003dea2…	"value1"	"value2"
6	ObjectId('6595eec8003dea2…	"value1"	"value1"
7	ObjectId('6595eec8003dea2…	"value1"	"value2"
8	ObjectId('6595eec8003dea2…	"value1"	"value2"
9	ObjectId('6595eec8003dea2…	"value1"	"value2"
10	ObjectId('6595eec8003dea2…	"value1"	"value1"

Рисунок 3.15 — Результат відпрацювання функції додавання 10 документів в MongoDB

В БД Google Firestore додалося 10 документів з відповідними полями, що вказує на те що функція додавання коректно працює з Google Firestore. Результат роботи операції додавання в хмарну базу даних Google Firestore наведена на рисунку 3.16:

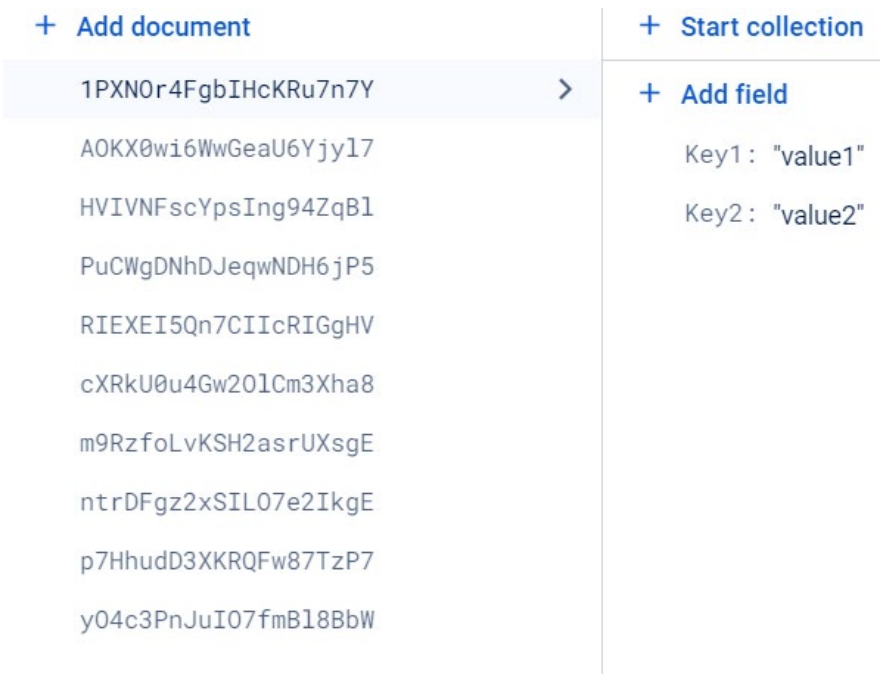


Рисунок 3.16 — Результат відпрацювання функції додавання 10 документів Google Firestore

Спочатку перевіримо коректність роботи функції отримання документів з колекції з пустим фільтром, що вказує на те що, повинні повернутися всі документи з колекції в БД В вікно фільтрів залишимо пустим. Вікно створення фільтру наведено на рисунку 3.17:

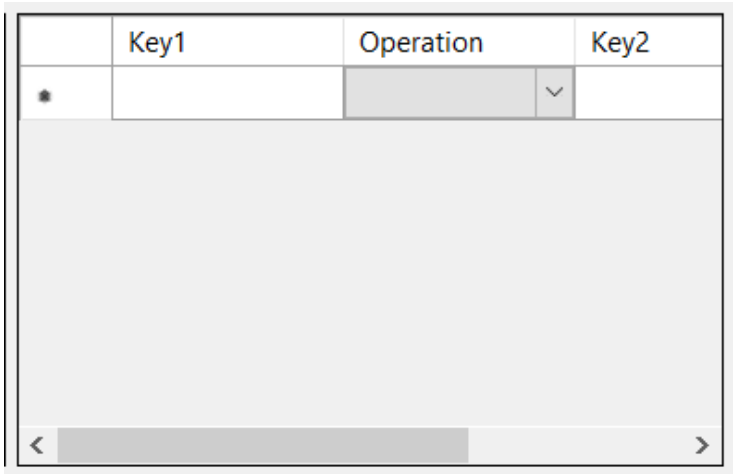


Рисунок 3.17 — Вікно створення фільтру

З БД mongoDB вдалося отримати 10 документів з відповідними полями, що вказує на те що функція отримання документів коректно працює з mongoDB. Результат роботи операції отримання документів з хмарної бази даних MongoDB наведена на рисунку 3.18:

Рисунок 3.18 — Результат відпрацювання функції отримання документів з MongoDB

З БД Google Firestore вдалося отримати 10 документів з відповідними полями, що вказує на те що функція отримання документів коректно працює з Google Firestore. Результат роботи операції отримання документів з хмарної бази даних Google Firestore наведена на рисунку 3.19:

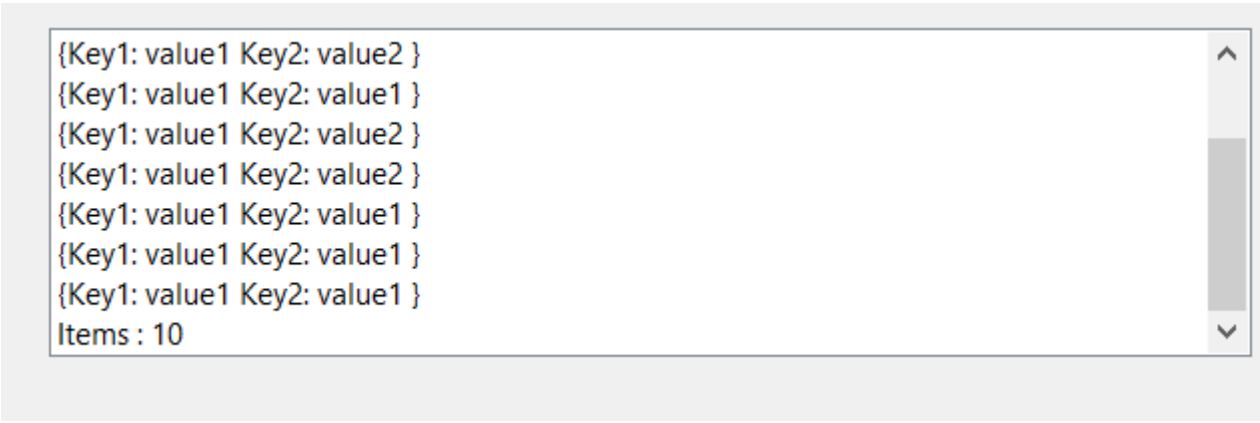


Рисунок 3.19 — Результат відпрацювання функції отримання документів з Google Firestore

Наступним кроком перевіримо коректність роботи функції отримання документів з колекції певним з фільтром, що вказує на те що, повинні повернутися ті документи які задовольняють умовам фільтру. В вікно фільтру за пишемо фільтр: Поле Key2 дорівнює value1. Створення фільтру наведено на рисунку 3.20:

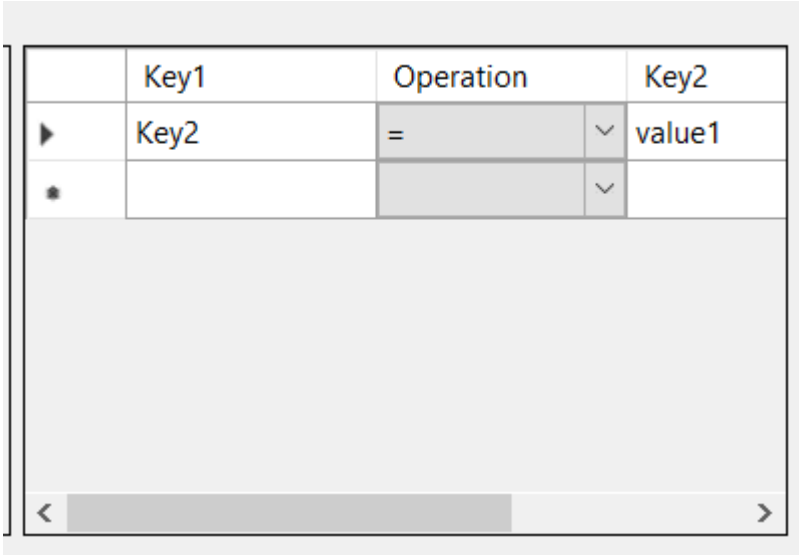


Рисунок 3.20 — Створення фільтру до документів

З БД mongoDB вдалося отримати 5 документів які задовольняють умовам фільтру, що вказує на те що функція отримання документів коректно працює з mongoDB. Результат роботи операції отримання відфільтрованих документів з хмарної бази даних MongoDB наведено на рисунку 3.21:

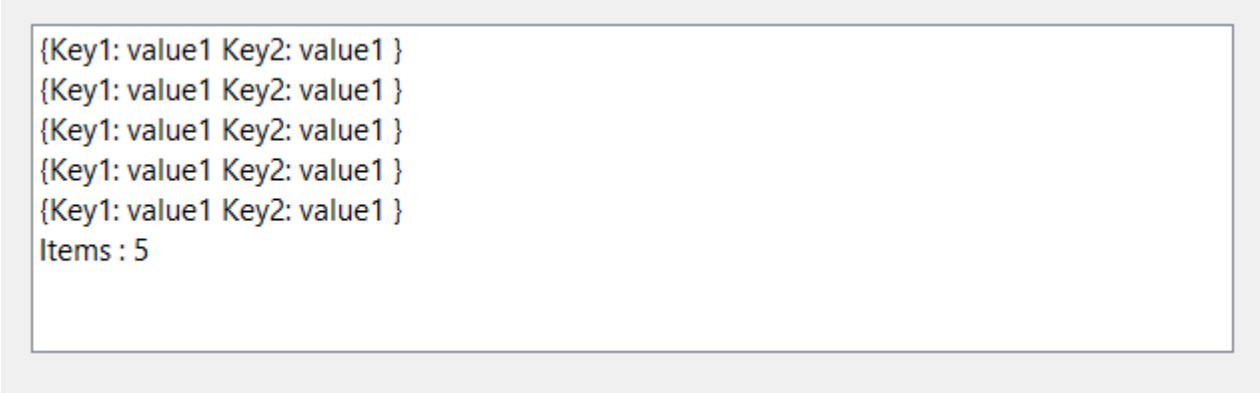


Рисунок 3.21 — Результат відпрацювання функції отримання документів з фільтром з MongoDB

З БД Google Firestore вдалося отримати 5 документів які задовольняють умовам фільтру, що вказує на те що функція отримання документів коректно працює з Google Firestore. Результат роботи операції отримання відфільтрованих документів з хмарної бази даних з Google Firestore наведено на рисунку 3.22:

Рисунок 3.22 — Результат відпрацювання функції отримання документів з фільтром з Google Firestore

Наступним кроком перевіримо коректність роботи функції оновлення документів в колекції певним з фільтром, що вказує на те що, повинні оновитися ті документи які задовольняють умовам фільтру. В вікно фільтру за пишемо фільтр: Поле Key2 дорівнює value1. А в вікно оновлення документу оновити поле Key2 новим значенням value3. Загальний вигляд вікон оновлення документу та фільтру наведено на рисунку 3.23:

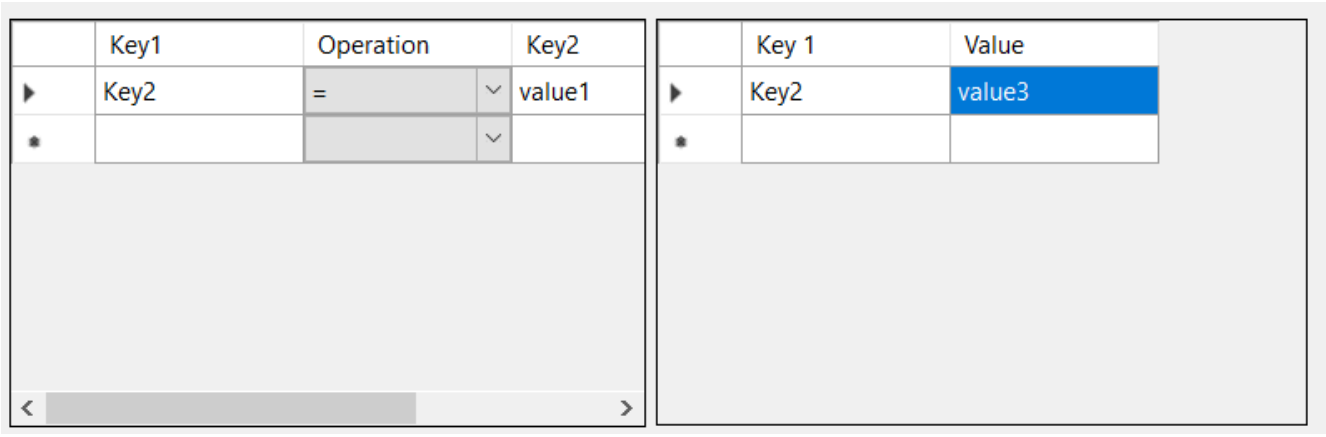


Рисунок 3.23 — Загальний вигляд оновлення документів в колекції за фільтром

В БД mongoDB вдалося оновити 5 документів які задовольняють умовам фільтру, що вказує на те що функція оновлення документів коректно працює з mongoDB. Результат роботи операції оновлення відфільтрованих документів в хмарній базі даних MongoDB наведено на рисунку 3.24:

123

	_id ObjectId	Key1 String	Key2 String
1	ObjectId('6595eec7003dea2…	"value1"	"value3"
2	ObjectId('6595eec7003dea2…	"value1"	"value3"
3	ObjectId('6595eec7003dea2…	"value1"	"value3"
4	ObjectId('6595eec7003dea2…	"value1"	"value2"
5	ObjectId('6595eec7003dea2…	"value1"	"value2"
6	ObjectId('6595eec8003dea2…	"value1"	"value3"
7	ObjectId('6595eec8003dea2…	"value1"	"value2"
8	ObjectId('6595eec8003dea2…	"value1"	"value2"
9	ObjectId('6595eec8003dea2…	"value1"	"value2"
10	ObjectId('6595eec8003dea2…	"value1"	"value3"

Рисунок 3.24 — Результат роботи функції оновлення за фільтром в MongoDB

В БД Google Firestore вдалося оновити 5 документів які задовольняють умовам фільтру, що вказує на те що функція оновлення документів коректно працює з Google Firestore. Результат роботи операції оновлення відфільтрованих документів в хмарній базі даних Google Firestore наведено на рисунку 3.25:

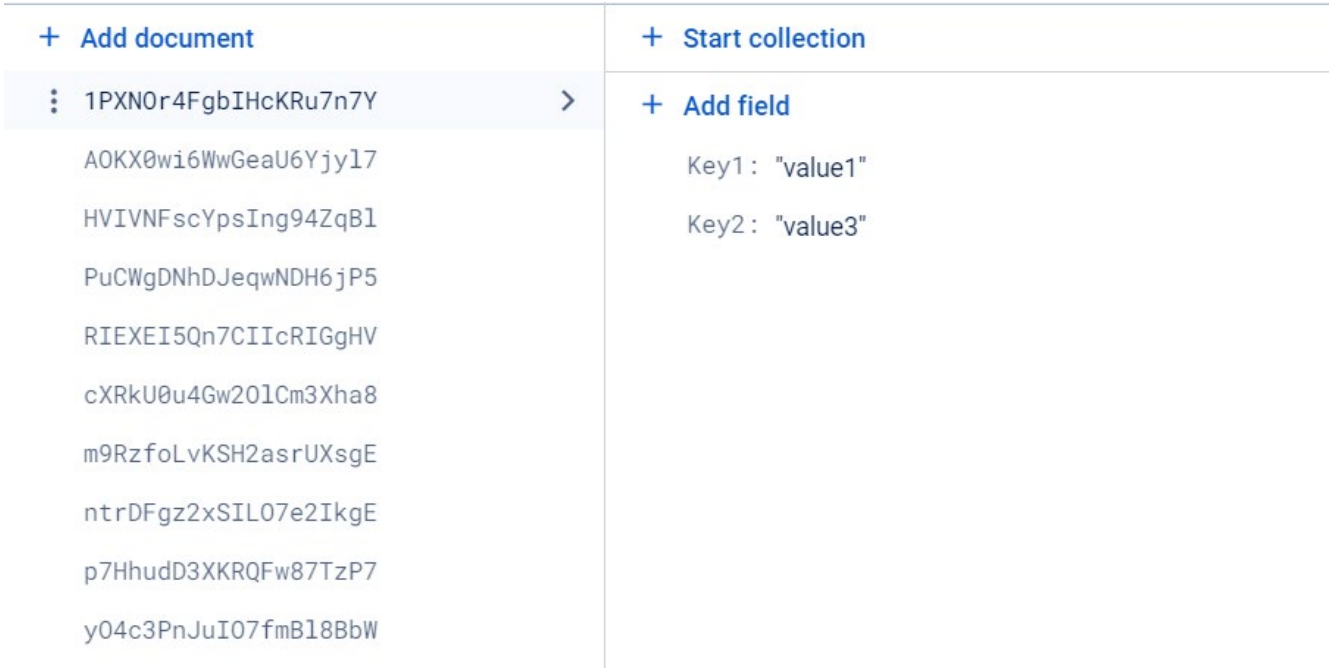


Рисунок 3.25 — Результат роботи функції оновлення за фільтром в Google Firestore

Наступним кроком перевіримо коректність роботи функції видалення документів в колекції з певним фільтром, що вказує на те що, повинні видалитись ті документи які задовольняють умовам фільтру. В вікно фільтру за пишемо фільтр: Поле Key2 дорівнює value2. Умова фільтру для видалення документів з баз даних наведено на рисунку 3.26:

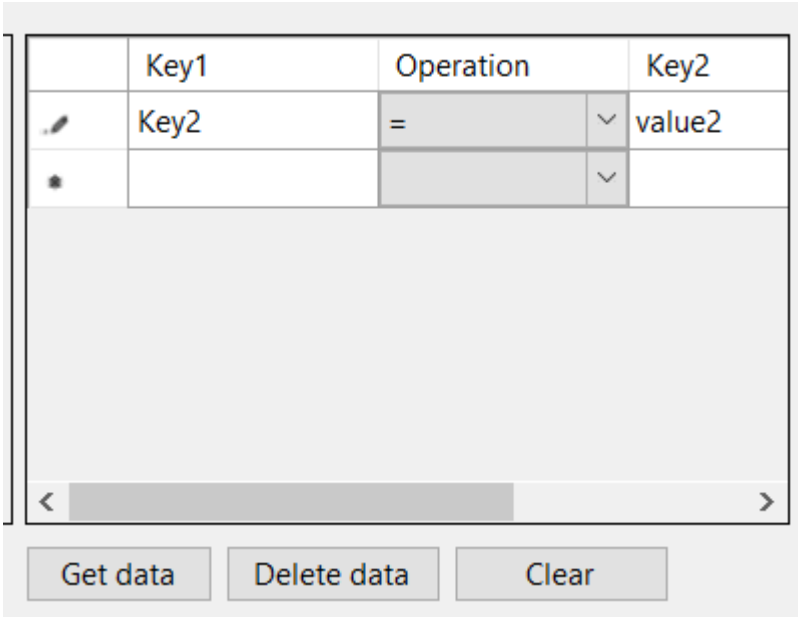
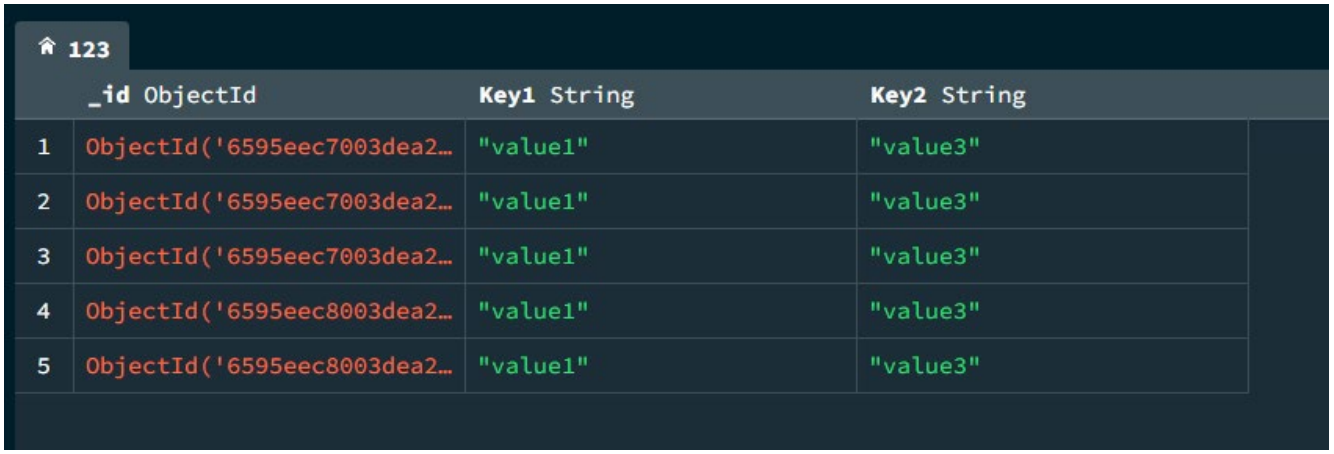


Рисунок 3.26 — Фільтр видалення документів

В БД mongoDB вдалося видалити 5 документів які задовольняють умовам фільтру, що вказує на те що функція видалення документів коректно працює з mongoDB. Результат роботи операції видалення відфільтрованих документів в хмарній базі даних MongoDB наведено на рисунку 3.27:


123

	_id ObjectId	Key1 String	Key2 String
1	ObjectId('6595eec7003dea2…	"value1"	"value3"
2	ObjectId('6595eec7003dea2…	"value1"	"value3"
3	ObjectId('6595eec7003dea2…	"value1"	"value3"
4	ObjectId('6595eec8003dea2…	"value1"	"value3"
5	ObjectId('6595eec8003dea2…	"value1"	"value3"

Рисунок 3.27 — Результат роботи функції видалення за фільтром в MongoDB

В БД Google Firestore вдалося видалити 5 документів які задовольняють умовам фільтру, що вказує на те що функція видалення документів коректно працює з Google Firestore. Результат роботи операції видалення відфільтрованих документів в хмарній базі даних Google Firestore наведено на рисунку 3.28:

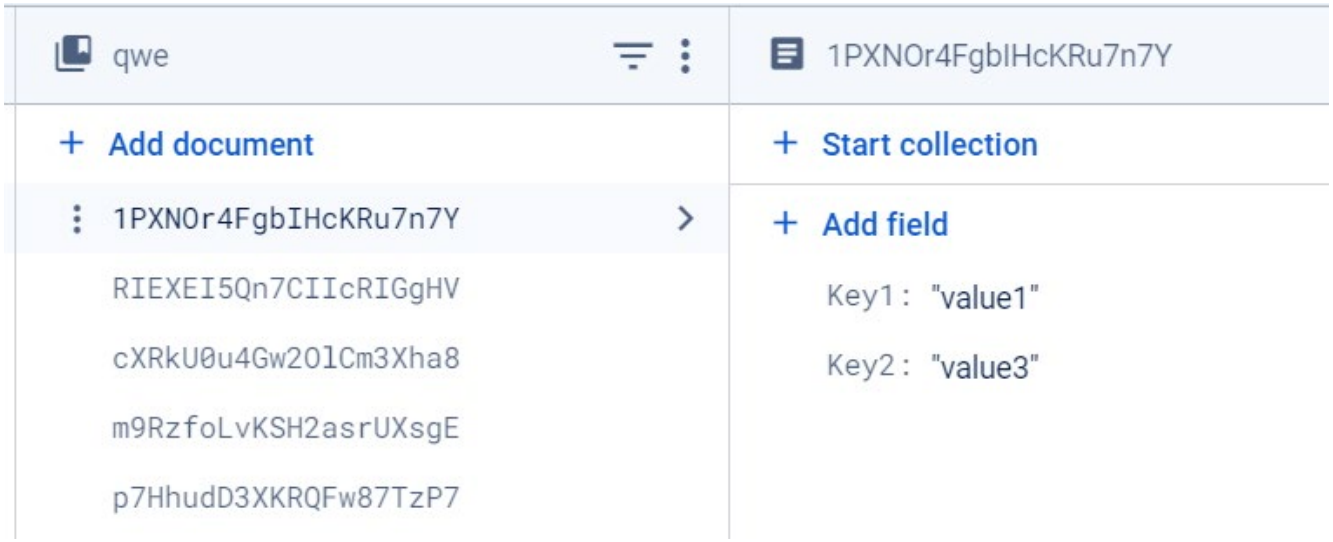


Рисунок 3.28 — Результат роботи функції оновлення за фільтром в Google Firestore

Висновки до розділу 3

Третій розділ містить у собі формалізацію алгоритму тестування додатку, опис мови програмування та інструментів які будуть використовуватися під час розробки, опис особливості роботи з бібліотеками для MongoDB та Google Firestore.

Також було проведене тестування програмної реалізації роботи з основними запитами MongoDB та Google Firestore:

- додавання документів до баз даних;
- отримання всіх документів з баз даних;
- отримання відфільтрованих документів;
- оновлення відфільтрованих документів;
- видалення відфільтрованих документів.

РОЗДІЛ 4 АНАЛІЗ ЕФЕКТИВНОСТІ ТА ПРОДУКТИВНОСТІ СИСТЕМ ДОБД

Для дослідження функціональних можливостей було проведення тестування баз даних з різною кількістю документів та розміром самих документів.

Кількість документів:

- 100 документів;
- 1000 документів;
- 10000 документів.

Розмір документів:

- 5 байт;
- 500 байт;
- 5000 байт.

Критерії які будуть оцінені:

- час додавання всіх документів;
- час отримання всіх документів;
- час видалення всіх документів;
- час оновлення всіх документів;
- час отримання документів використовуючи фільтр документів;
- час оновлення документів використовуючи фільтр документів;
- час видалення документів використовуючи фільтр документів;
- Весь час наведений в мілісекундах.

4.1 Дослідження роботи під час опрацювання 100 документів

Під час дослідження часу виконання операцій з базами даних ми отримали часовий результат який наведений в таблиці 4.1:

Таблиця 4.1 – Час виконання операцій

		5	500	5000
add	mongo	6934	7392	7856
	google	10337	9083	9504
get	mongo	990	1556	1232
	google	551	647	682
delete	mongo	156	198	176
	google	399	331	347
update	mongo	188	156	208
	google	62	63	41
get with condition	mongo	408	749	433
	google	238	460	399
update with condition	mongo	189	216	179
	google	80	79	84
delete with condition	mongo	177	207	192
	google	156	182	178

З отриманих даних було складено порівняльні графіки.

Графік затраченого часу на додавання 100 документів до БД представлено на малюнку 4.1:

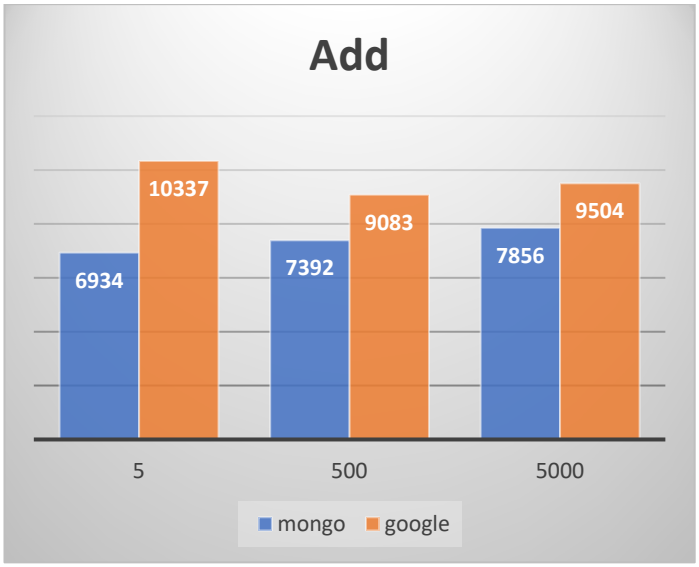


Рисунок 4.1 — Графік порівняння затраченого часу на додавання документів до БД

З отриманого графіку можна сказати що операції додавання документів до бази даних MongoDB займає менше часу ніж додавання до Google Firestore.

Графік затраченого часу на отримання 100 документів з БД представлено на малюнку 4.2:

Рисунок 4.2 — Графік порівняння затраченого часу на отримання всіх документів з БД

З отриманого графіку можна сказати що операції отримання документів з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік затраченого часу на видалення всіх документів з БД наведено на рисунку 4.3:

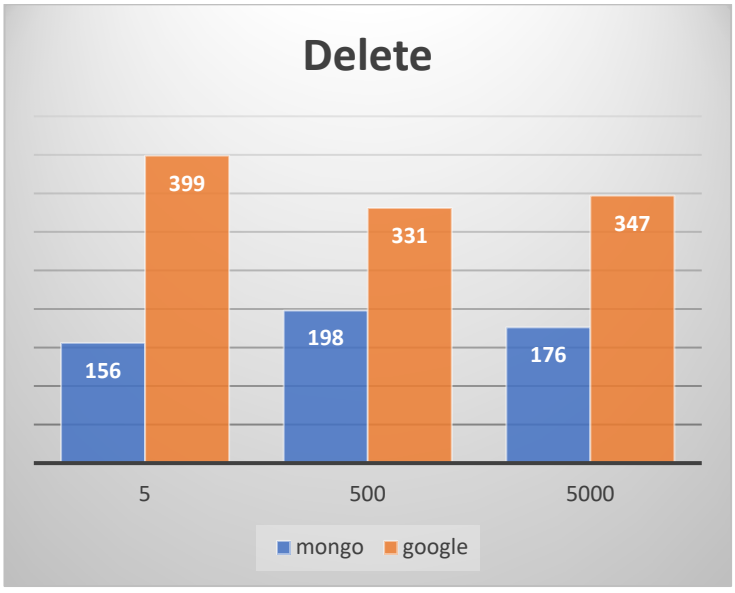


Рисунок 4.3 — Графік порівняння затраченого часу на видалення всіх документів до БД

З отриманого графіку можна сказати що операції видалення документів з бази даних MongoDB займає менше часу ніж видалення з Google Firestore.

Графік затраченого часу на оновлення всіх документів в БД наведено на рисунку 4.4:

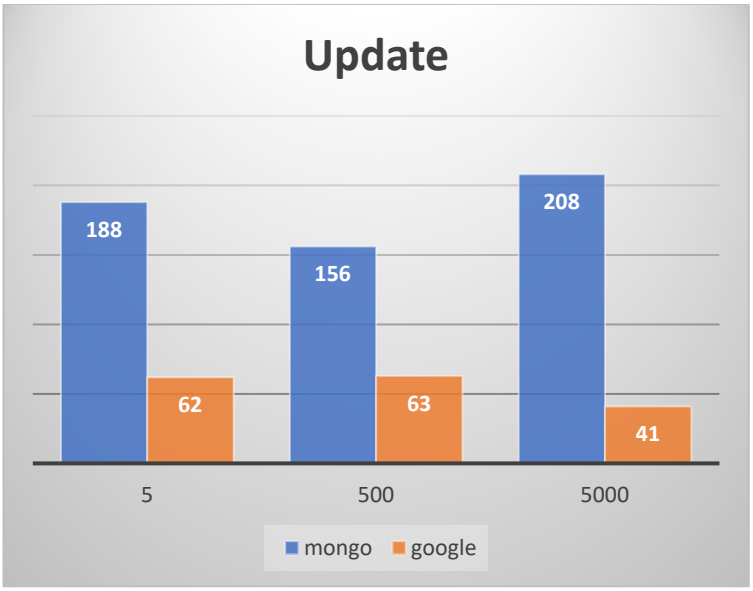


Рисунок 4.4 — Графік порівняння затраченого часу на оновлення всіх документів до БД

З отриманого графіку можна сказати що операції оновлення документів в базі даних MongoDB займає більше часу ніж оновлення в Google Firestore.

Графік затраченого часу на отримання документів за фільтром з БД наведено на рисунку 4.5.

З отриманого графіку можна сказати що операції отримання документів за фільтром з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік затраченого часу на оновлення документів за фільтром з БД наведено на рисунку 4.6:

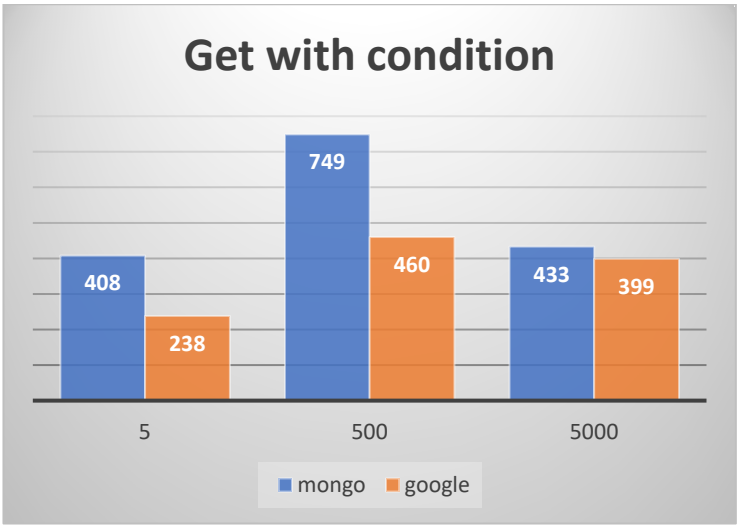


Рисунок 4.5 — Графік порівняння затраченого часу на отримання документів за фільтром з БД

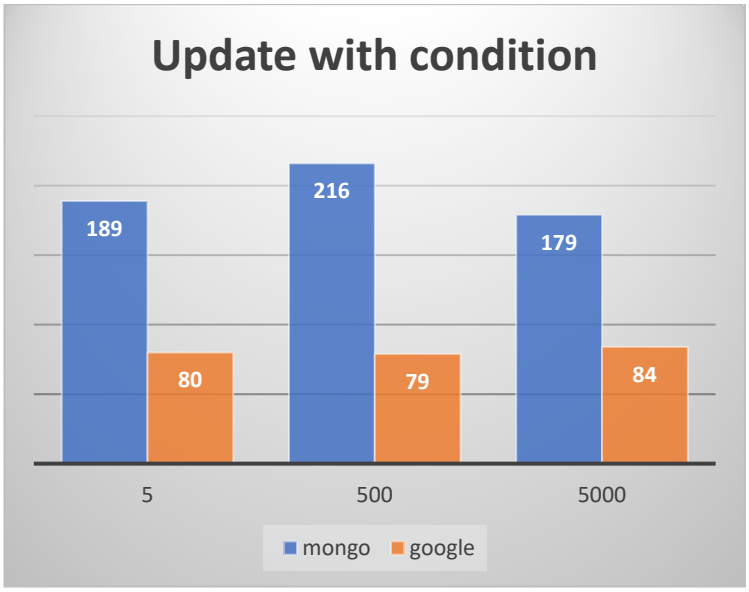


Рисунок 4.6 — Графік порівняння затраченого часу на оновлення документів за фільтром в БД

З отриманого графіку можна сказати що операції оновлення документів за фільтром в базі даних MongoDB займає більше часу ніж оновлення з Google Firestore.

Графік затраченого часу на видалення документів за фільтром в БД наведено на рисунку 4.7:

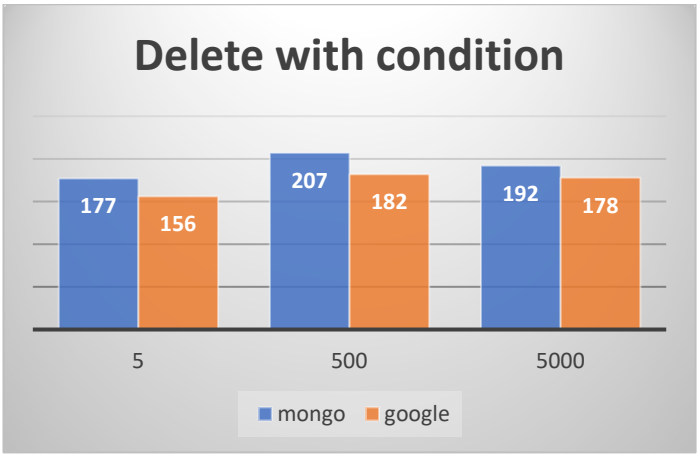


Рисунок 4.7 — Графік порівняння затраченого часу на видалення документів за фільтром з БД

З отриманого графіку можна сказати що операції видалення документів за фільтром з бази даних MongoDB займає більше часу ніж видалення з Google Firestore.

4.2 Дослідження роботи під час опрацювання 1000 документів

Під час дослідження часу виконання операцій з базами даних ми отримали часовий результат який наведений в таблиці 2.

Таблиця 4.2 - Час виконання операцій

		5	500	5000
add	mongo	69588	71599	72874
	google	79884	78651	79273
get	mongo	10691	11954	11162
	google	8226	8446	8420
delete	mongo	404	386	372
	google	452	536	595
update	mongo	2532	2401	2275
	google	854	885	914
get with condition	mongo	5779	5399	5157
	google	3470	3348	4033
update with condition	mongo	2487	2369	2528
	google	1043	1014	1137
delete with condition	mongo	2582	2431	2851
	google	1071	1040	1015

Графік затраченого часу на додавання 1000 документів до БД представлено на малюнку 4.8:

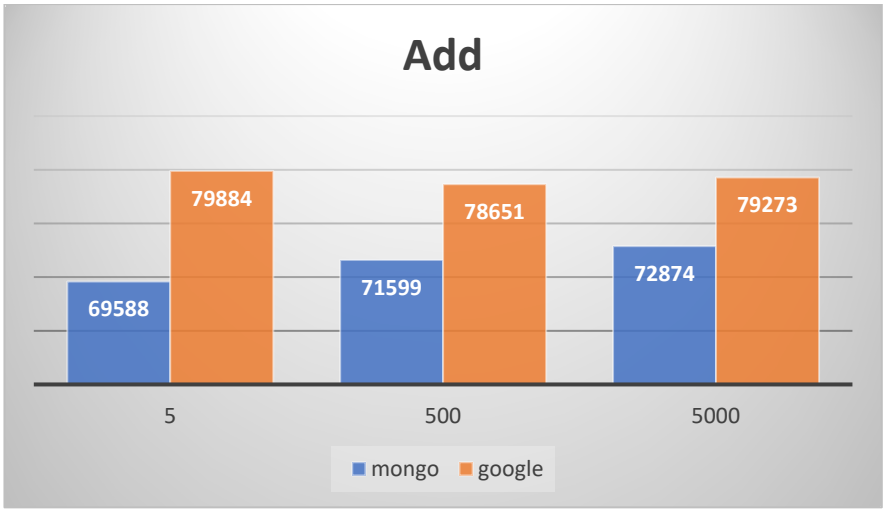


Рисунок 4.8 — Графік порівняння затраченого часу на додавання документів до БД

З отриманого графіку можна сказати що операції додавання документів до бази даних MongoDB займає менше часу ніж додавання до Google Firestore.

Графік затраченого часу на отримання 1000 документів з БД представлено на малюнку 4.9:

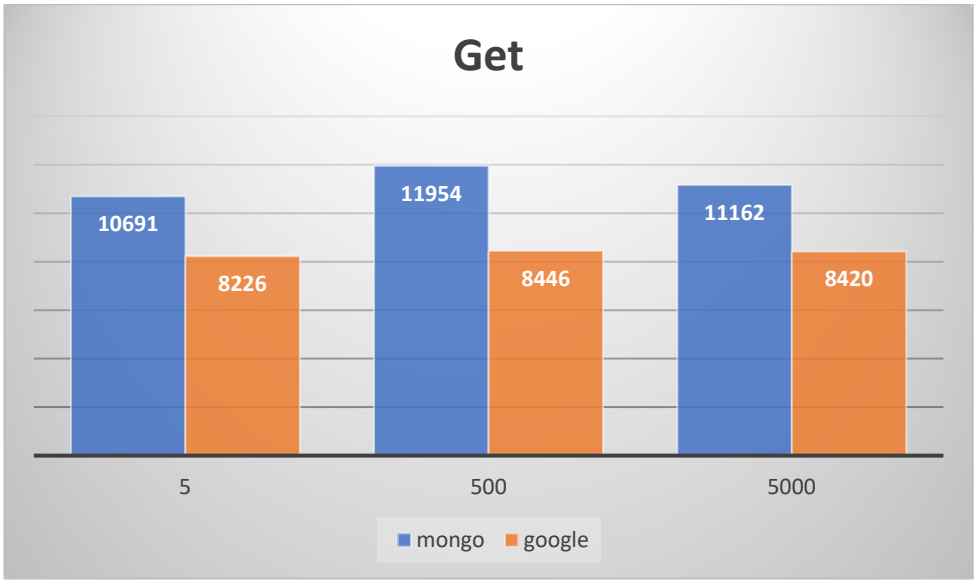


Рисунок 4.9 — Графік порівняння затраченого часу на отримання всіх документів з БД

З отриманого графіку можна сказати що операції отримання документів з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік затраченого часу на видалення всіх документів з БД наведено на рисунку 4.10:

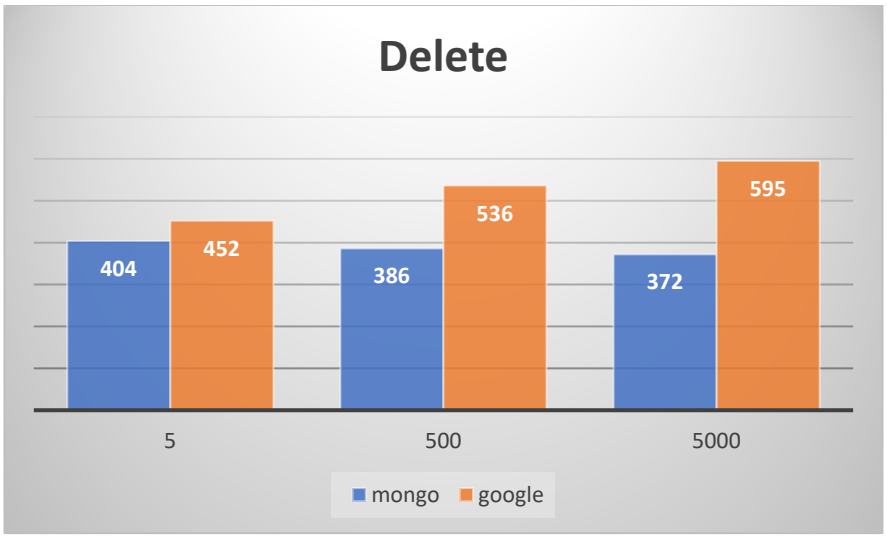


Рисунок 4.10 — Графік порівняння затраченого часу на видалення всіх документів до БД

З отриманого графіку можна сказати що операції видалення документів з бази даних MongoDB займає менше часу ніж видалення з Google Firestore.

Графік затраченого часу на оновлення всіх документів в БД наведено на рисунку 4.11:

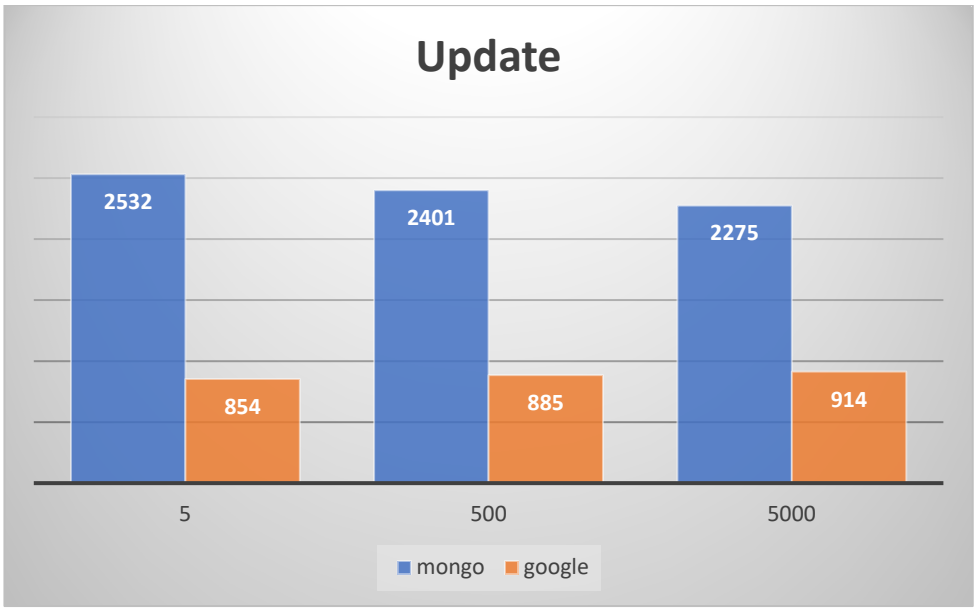


Рисунок 4.11 — Графік порівняння затраченого часу на оновлення всіх документів до БД

З отриманого графіку можна сказати що операції оновлення документів в базі даних MongoDB займає більше часу ніж оновлення в Google Firestore.

Графік затраченого часу на отримання документів за фільтром з БД наведено на рисунку 4.12

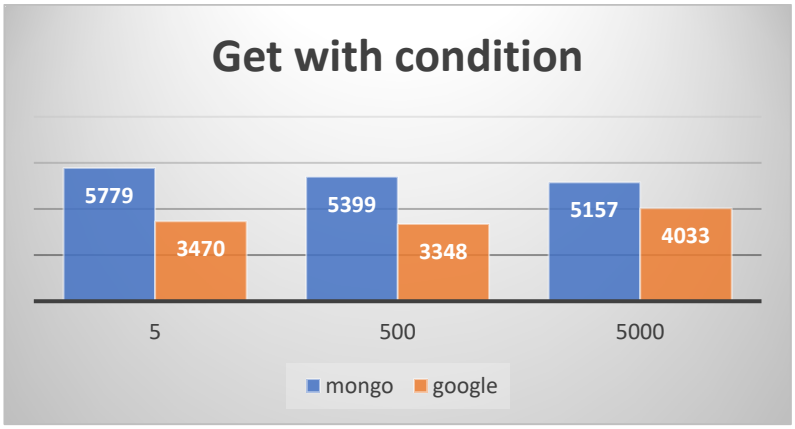


Рисунок 4.12 — Графік порівняння затраченого часу на отримання документів за фільтром з БД

З отриманого графіку можна сказати що операції отримання документів за фільтром з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік затраченого часу на оновлення документів за фільтром з БД наведено на рисунку 4.13:

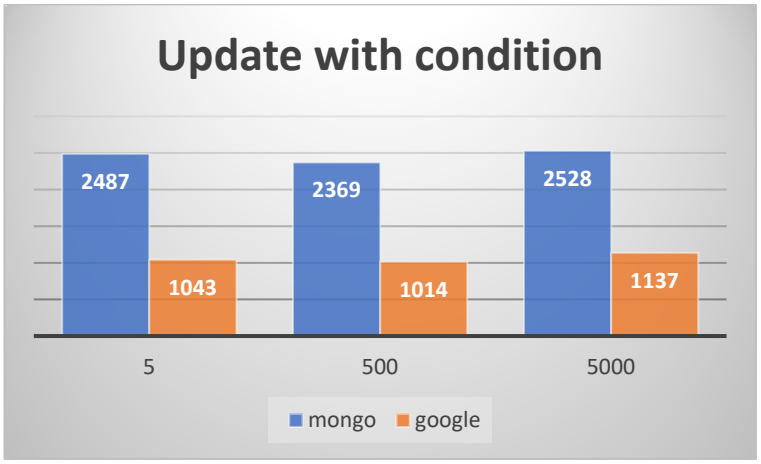


Рисунок 4.13 — Графік порівняння затраченого часу на оновлення документів за фільтром в БД

З отриманого графіку можна сказати що операції оновлення документів за фільтром в базі даних MongoDB займає більше часу ніж оновлення з Google Firestore.

Графік затраченого часу на видалення документів за фільтром в БД наведено на рисунку 4.14:

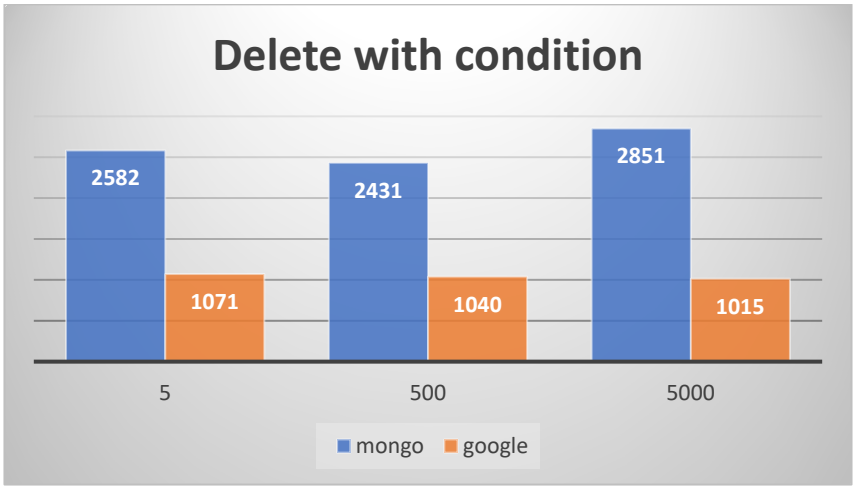


Рисунок 4.14 — Графік порівняння затраченого часу на видалення документів за фільтром з БД

З отриманого графіку можна сказати що операції видалення документів за фільтром з бази даних MongoDB займає більше часу ніж видалення з Google Firestore.

4.3 Дослідження роботи під час опрацювання 10000 документів

Під час дослідження часу виконання операцій з базами даних ми отримали часовий результат який наведений в таблиці 3.

Таблиця 4.3 - Час виконання операцій

		5	500	5000
add	mongo	533172	532730	543641
	google	644268	635606	709159
get	mongo	4444	7600	4025
	google	3369	3646	3873
delete	mongo	1026	839	731
	google	1412	1484	1234
update	mongo	869	1105	1225
	google	198	235	218

Продовження таблиці 4.3

		5	500	5000
get with condition	mongo	856	771	3480
	google	610	755	728
update with condition	mongo	1087	842	580
	google	528	541	554
delete with condition	mongo	230	831	948
	google	403	487	547

Графік затраченого часу на додавання 10000 документів до БД представлено на малюнку 4.15:

Рисунок 4.15 — Графік порівняння затраченого часу на додавання документів до БД

З отриманого графіку можна сказати що операції додавання документів до бази даних MongoDB займає менше часу ніж додавання до Google Firestore.

Графік затраченого часу на отримання 1000 документів з БД представлено на малюнку 4.16:

З отриманого графіку можна сказати що операції отримання документів з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік затраченого часу на видалення всіх документів з БД наведено на рисунку 4.17:

Рисунок 4.16 — Графік порівняння затраченого часу на отримання всіх документів з БД

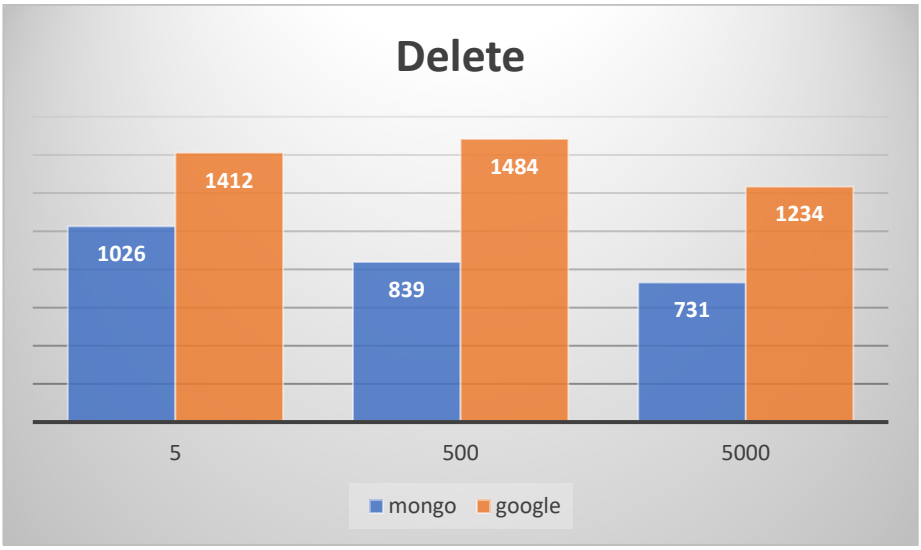


Рисунок 4.17 — Графік порівняння затраченого часу на видалення всіх документів до БД

З отриманого графіку можна сказати що операції видалення документів з бази даних MongoDB займає менше часу ніж видалення з Google Firestore.

Графік затраченого часу на оновлення всіх документів в БД наведено на рисунку 4.18:

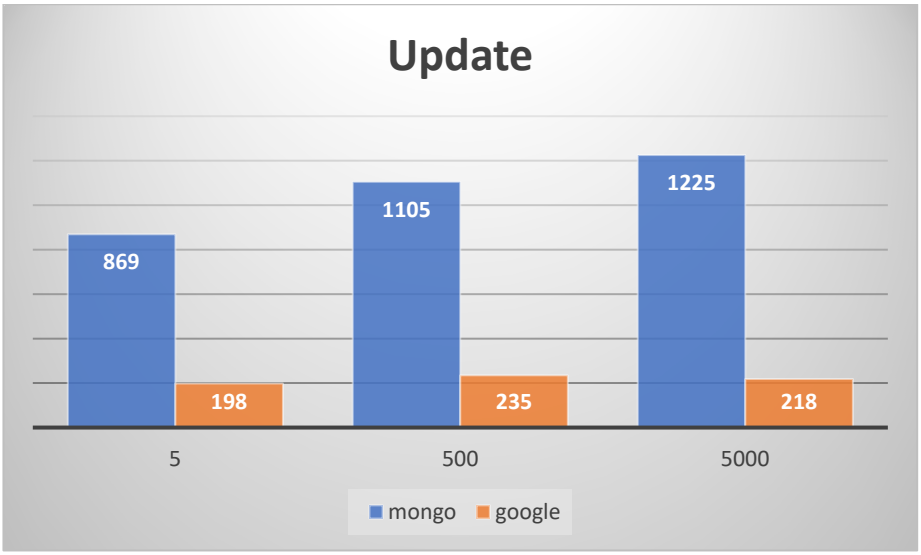


Рисунок 4.18 — Графік порівняння затраченого часу на оновлення всіх документів до БД

З отриманого графіку можна сказати що операції оновлення документів в базі даних MongoDB займає більше часу ніж оновлення в Google Firestore.

Графік затраченого часу на отримання документів за фільтром з БД наведено на рисунку 4.19:

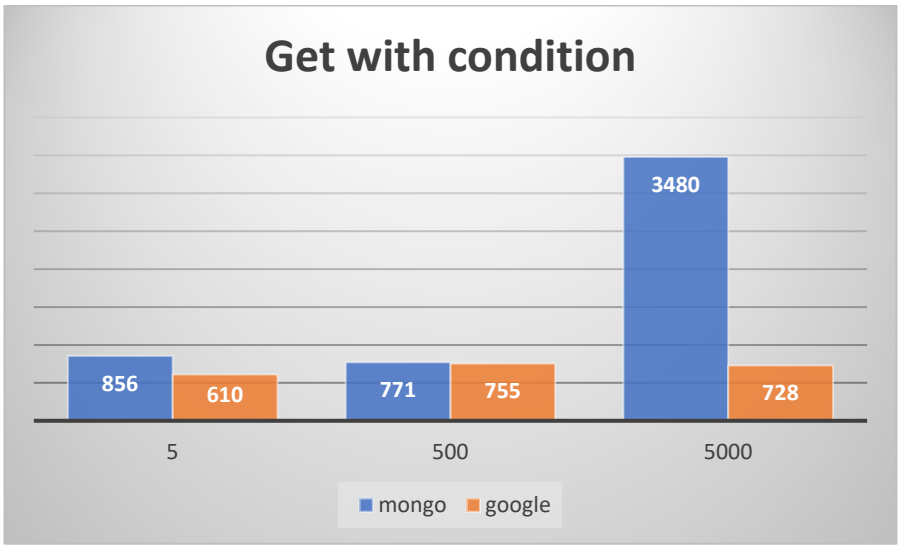


Рисунок 4.19 — Графік порівняння затраченого часу на отримання документів за фільтром з БД

З отриманого графіку можна сказати що операції отримання документів за фільтром з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік затраченого часу на оновлення документів за фільтром з БД наведено на рисунку 4.20:

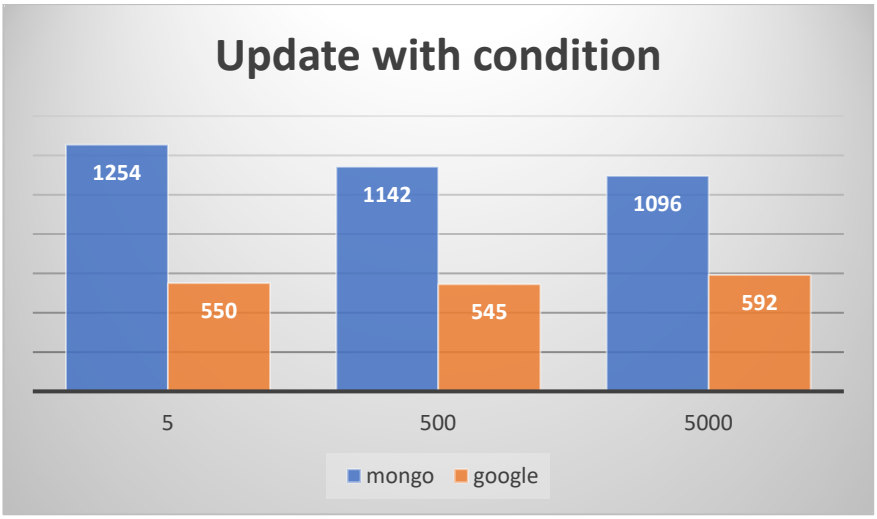


Рисунок 4.20 — Графік порівняння затраченого часу на оновлення документів за фільтром в БД

З отриманого графіку можна сказати що операції оновлення документів за фільтром в базі даних MongoDB займає більше часу ніж оновлення з Google Firestore.

Графік затраченого часу на видалення документів за фільтром в БД наведено на рисунку 4.21:

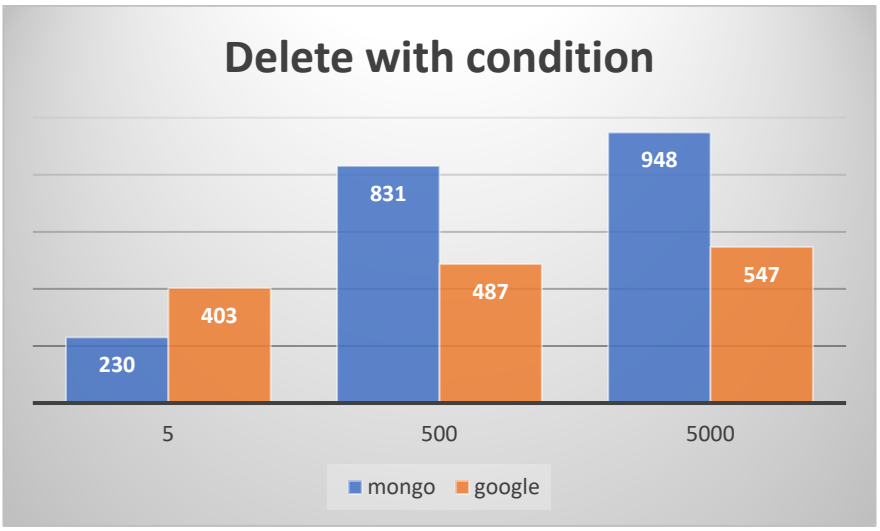


Рисунок 4.21 — Графік порівняння затраченого часу на видалення документів за фільтром з БД

З отриманого графіку можна сказати що операції видалення документів за фільтром з бази даних MongoDB займає більше часу ніж видалення з Google Firestore.

4.2 Висновки за розділом 4

Проведене часове дослідження роботи документ-орієнтованих баз даних (ДОБД) в контексті виконання операцій CRUD (створення, читання, оновлення, видалення) над даними виявило важливі аспекти та розкрило різницю у часовій ефективності між обраними системами. Підсумовуючи отримані дані можна привести їх до середнього значення. Середні значення представлено в таблиці 4.

Таблиця 4.4 – Середній час виконання операцій

		5	500	5000
add	mongo	203231	203907	208124
	google	244830	241113	265979
get	mongo	5375	7037	5473
	google	4049	4246	4325
delete	mongo	529	474	426
	google	754	784	725
update	mongo	1196	1221	1236
	google	371	394	391
get with condition	mongo	2348	2306	3023
	google	1439	1521	1720
update with condition	mongo	1254	1142	1096
	google	550	545	592
delete with condition	mongo	996	1156	1330
	google	543	570	580

З отриманих даних було складено порівняльні графіки:

Середній графік затраченого часу на додавання документів до БД наведено рисунку 4.22:

Рисунок 4.22 — Графік порівняння затраченого часу на додавання документів до БД

З отриманого графіку можна сказати, що в середньому операції додавання документів до бази даних MongoDB займає менше часу ніж додавання до Google Firestore.

Графік середнього затраченого часу на отримання документів з БД представлено на малюнку 4.23:

Рисунок 4.23 — Графік порівняння затраченого часу на отримання всіх документів з БД

З отриманого графіку можна сказати, що в середньому операції отримання документів з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік середнього затраченого часу на видалення документів з БД наведено на рисунку 4.24:

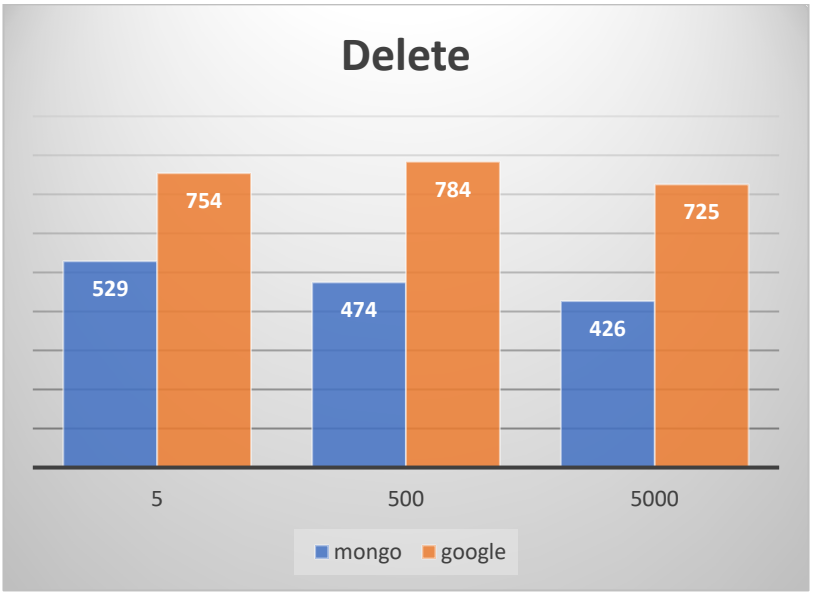


Рисунок 4.24 — Графік порівняння затраченого часу на видалення всіх документів до БД

З отриманого графіку можна сказати , що в середньому операції видалення документів з бази даних MongoDB займає менше часу ніж видалення з Google Firestore.

Графік середнього затраченого часу на оновлення документів в БД наведено на рисунку 4.25:

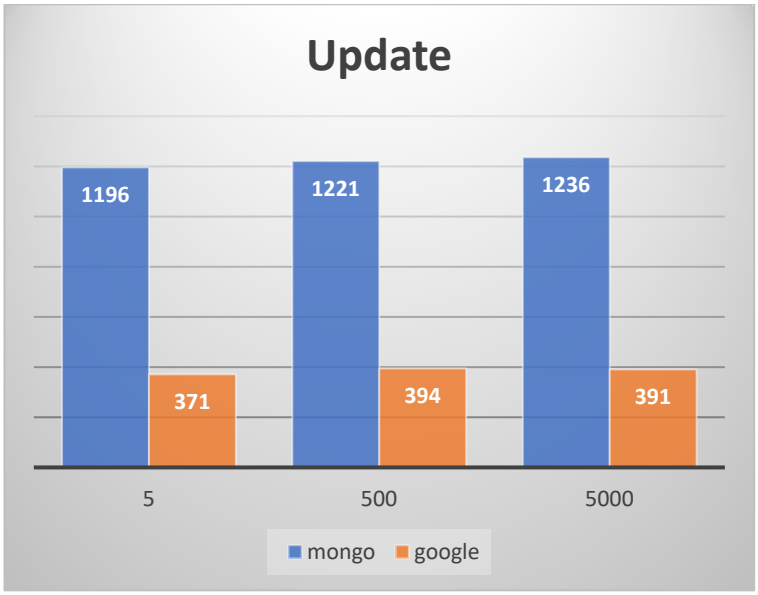


Рисунок 4.25 — Графік порівняння затраченого часу на оновлення всіх документів до БД

З отриманого графіку можна сказати, що в середньому операції оновлення документів в базі даних MongoDB займає більше часу ніж оновлення в Google Firestore.

Графік затраченого часу на отримання документів за фільтром з БД наведено на рисунку 4.26

Рисунок 4.26 — Графік порівняння затраченого часу на отримання документів за фільтром з БД

З отриманого графіку можна сказати, що в середньому операції отримання документів за фільтром з бази даних MongoDB займає більше часу ніж отримання з Google Firestore.

Графік затраченого часу на оновлення документів за фільтром з БД наведено на рисунку 4.27:

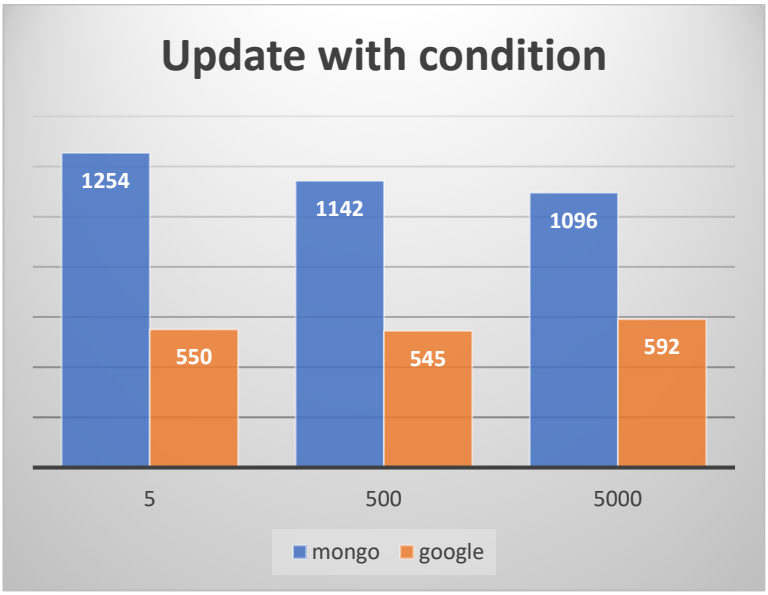


Рисунок 4.27 — Графік порівняння затраченого часу на оновлення документів за фільтром в БД

З отриманого графіку можна сказати, що в середньому операції оновлення документів за фільтром в базі даних MongoDB займає більше часу ніж оновлення з Google Firestore.

Графік затраченого часу на видалення документів за фільтром в БД наведено на рисунку 4.28:

З отриманого графіку можна сказати, що в середньому операції видалення документів за фільтром з бази даних MongoDB займає більше часу ніж видалення з Google Firestore.

З отриманих результатів можна сказати що MongoDB краще використовувати коли треба швидко додавати та видаляти великі обсяги даних, в той час як Google Forestore краще використовувати в тих випадках коли потрібно швидке отримання та оновлення даних.

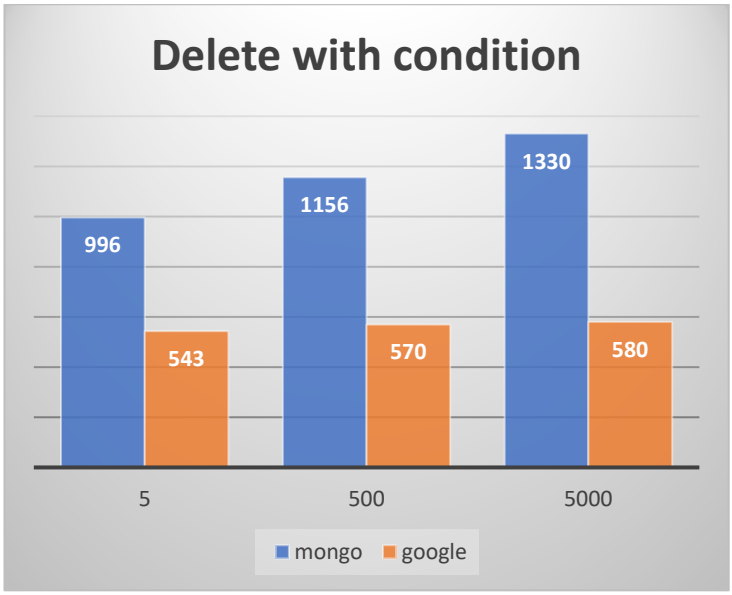


Рисунок 4.28 — Графік порівняння затраченого часу на видалення документів за фільтром з БД

ЗАГАЛЬНИЙ ВИСНОВОК

Під час виконання дипломної роботи було досліджено функціональні можливості документ-орієнтованих баз даних MongoDB та Google Firestore. Було розглянуто основні поняття на яких побудовані документ-орієнтовані бази даних.

Під час виконання роботи було створено додаток який надавав можливість користувачу працювати з даними в хмарних базах даних. Розглянуто нюанси з використання кожної з баз даних.

Було приділено багато уваги то тестування додатку та заміру часу виконання основних операцій з документ-орієнтованими базами даних, а саме:

- додавання документів до баз даних;
- отримання документів з баз даних;
- оновлення документів в базах даних;
- видалення документів з баз даних.

Було проведено порівняльний аналіз часу виконання кожної з основних операцій та приведені рекомендації яку з баз даних краще використовувати в конкретній ситуації.

Також було розглянуті бібліотеки які використовувались під час виконання роботи.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. І.Б. Швороб. (2016). НОВИЙ ПІДХІД ДО ЗБЕРЕЖЕННЯ СЛАБОСТРУКТУРОВАНИХ МЕДИЧНИХ ДАНИХ. Науковий вісник НЛТУ України, 26(4), 382-390. https://nv.nltu.edu.ua/Archive/2016/26_4/61.pdf.

2. Gillis A. S., Botelho B. What is MongoDB? Features and how it works – TechTarget Definition. Data Management. URL: https://www.techtarget.com/searchdatamanagement/definition/MongoDB (date of access: 15.12.2023).

3. MongoDB Manual $Or. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/reference/operator/query/or/ (date of access: 15.12.2023).

4. MongoDB Manual updateMany. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/reference/method/db.collection.updateMany/ (date of access: 15.12.2023).

5. MongoDB manual $and. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/reference/operator/query/and/ (date of access: 15.12.2023).

6. MongoDB Manual deleteMany. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/reference/method/db.collection.deleteMany/ (date of access: 15.12.2023).

7. Pauly S. Why I switched away from google firestore – and don't regret it. Spencer Pauly | Home. URL: https://spencerpauly.com/tech/why-i-switched-away-from-google-firestore/ (date of access: 10.12.2023).

8. A tour of C# - Overview - C#. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (date of access: 10.12.2023).

9. Програмування на C#: основні характеристики та плюси мови. FoxmindEd.

URL: https://foxminded.ua/prohramuvannia-na-si/ (date of access: 15.12.2023).

10. Quick start C#/.NET. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/drivers/csharp/current/quick-start/#std-label-csharp-quickstart (date of access: 15.12.2023).

11. MongoDB Manual insertOne. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/reference/method/db.collection.insertOne/ (date of access: 15.12.2023).

12. MongoDB Manual find. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/reference/method/db.collection.find/ (date of access: 15.12.2023).

13. Get started with cloud firestore | firebase. Firebase. URL: https://firebase.google.com/docs/firestore/quickstart?hl=en (date of access: 20.12.2023).

14. Add data to cloud firestore | firebase. Firebase. URL: https://firebase.google.com/docs/firestore/manage-data/add-data?hl=en (date of access: 20.12.2023).

15. Get data with cloud firestore | firebase. Firebase. URL: https://firebase.google.com/docs/firestore/query-data/get-data?hl=en (date of access: 20.12.2023).

16. Delete data from cloud firestore | firebase. Firebase. URL: https://firebase.google.com/docs/firestore/manage-data/delete-data?hl=en (date of access: 20.12.2023).

ДОДАТОК А

Технічно завдання

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«ДОСЛІДЖЕННЯ ФУНКІОНАЛЬНИХ МОЖЛИВОСТЕЙ ДОКУМЕНТНООРІЄНТОВАНИХ БАЗ ДАНИХ MONGODB ТА GOOGLE FIRESTORE»
Технічне завдання
44165850.1351-01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Максим ПЛЯШКО
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО
44165850.1351-01

«ДОСЛІДЖЕННЯ ФУНКІОНАЛЬНИХ МОЖЛИВОСТЕЙ ДОКУМЕНТНООРІЄНТОВАНИХ БАЗ ДАНИХ MONGODB ТА GOOGLE FIRESTORE»

Технічне завдання
44165850.1351-01
Листів 15

2024

44165850.1351-01
ЗМІСТ

ВСТУП

В сучасному світі обробка та збереження даних стала однією з найважливіших складових інформаційних технологій. З метою забезпечення ефективного доступу до даних, швидкості їх обробки і збереження відповідних обсягів інформації, багато організацій обирають різноманітні системи управління базами даних (СУБД). Однією з інноваційних альтернатив традиційним реляційним базам даних є документ-орієнтовані бази даних.

У даному дослідженні ми зосередимо увагу на порівнянні та аналізі функціональних можливостей двох популярних документ-орієнтованих СУБД - MongoDB та Google Firestore. MongoDB - це відкрита, масштабована та гнучка база даних, яка базується на моделі документації JSON і здатна працювати з великою кількістю структурованих та неструктурованих даних. З іншого боку, Google Firestore - це клауд-сервіс від Google, який пропонує документно-орієнтовану базу даних з високою доступністю та масштабованістю.

Це дослідження розглядає основні можливості обох систем, їхні переваги та недоліки, а також порівнює їх з точки зору продуктивності, масштабованості, безпеки та інших аспектів. Ми також дослідимо сценарії, в яких кожна з цих СУБД може бути найкращим вибором, і проаналізуємо практичні приклади їх використання.

Основною метою цього дослідження є допомога організаціям та розробникам вибрати оптимальну документ-орієнтовану базу даних для своїх проектів, враховуючи їхні потреби та вимоги. Ми спробуємо відповісти на питання про те, яка з цих систем краще підходить для конкретних завдань та сценаріїв використання, та які переваги вони можуть принести користувачам у світі сучасних даних та інформаційних технологій.

44165850.1351-01

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ ректора Українського державного університету науки і технології Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» №1196 ст від 05.12. 2022 року.

Тема проекту: «Дослідження функціональних можливостей документ-орієнтованих баз даних MongoDB та Google Firestore».

Керівник дипломного проекту: Іванов О. П.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Ця розробка спрямована на проведення аналізу та порівняння функціональних можливостей двох популярних документ-орієнтованих баз даних - MongoDB та Google Firestore. Головною метою є дослідження та оцінка можливостей цих баз даних з метою визначення їхньої придатності для конкретних проектів та завдань.

Ця робота покликана виявити переваги та недоліки обох систем у таких аспектах, як продуктивність, масштабованість, безпека, підтримка операцій з даними та інші функціональність. Результати дослідження допоможуть користувачам обгрунтовано вибрати оптимальну документно-орієнтовану базу даних для їхніх проектів, враховуючи конкретні потреби та вимоги.

Крім того, ця робота спрямована на розгляд конкретних сценаріїв використання MongoDB та Google Firestore та надання практичних рекомендацій для їхнього успішного впровадження. Результати дослідження можуть бути корисними для організацій та розробників, які розглядають можливості використання документ-орієнтованих баз даних у своїх проектах.

2.1 Функціональне призначення розробки

Основні аспекти функціонального призначення розробки:

- проведення аналізу функціональних можливостей – основною функцією розробки є докладний аналіз функціональних можливостей обох документно-орієнтованих баз даних, зокрема MongoDB і Google Firestore. Це включає в себе вивчення їхніх можливостей для збереження, пошуку, операцій з даними, масштабованості та інших аспектів;
- порівняння функцій – розробка дозволить порівняти функції і можливості обох систем, визначити переваги та недоліки кожної з них та надати об'єктивну оцінку їхнього потенціалу;
- вивчення практичних застосувань – розробка також буде спрямована на вивчення практичних сценаріїв використання MongoDB та Google Firestore

в різних областях, включаючи розробку веб-додатків, мобільних додатків, аналітичних систем та інших проектів;

– надання рекомендацій – результати розробки допоможуть надати рекомендації організаціям та розробникам щодо вибору найкращої документно-орієнтованої бази даних відповідно до їхніх потреб і завдань.

Висновки та розробка документації – розробка завершиться формулюванням висновків та підготовкою документації, в якій будуть представлені результати дослідження та рекомендації для майбутнього використання MongoDB та Google Firestore.

2.2 Експлуатаційне призначення розробки:

Основні аспекти експлуатаційного призначення розробки:

– оцінка і вибір бази даних – результати дослідження допоможуть організаціям і розробникам визначити, яка з документно-орієнтованих баз даних, MongoDB або Google Firestore, найкраще відповідає їхнім потребам і завданням;
– планування розробки проектів – отримані знання про функціональні можливості баз даних допоможуть розробникам планувати та реалізовувати проекти, які вимагають використання документно-орієнтованих баз даних, забезпечуючи оптимальне використання їхніх функцій;
– оптимізація даних – знання про можливості MongoDB та Google Firestore дозволить оптимізувати структуру та обробку даних в проектах, що використовують ці бази даних;
– підтримка та розвиток проектів – розробка надає можливість підтримувати та розвивати існуючі проекти, використовуючи документно-орієнтовані бази даних для задоволення змінних потреб користувачів;

– підготовка документації – експлуатаційне призначення включає в себе підготовку документації, яка описує рекомендації, кращі практики та результати дослідження, щоб мати відповідні ресурси для майбутніх проектів та потреб користувачів.

У підсумку, експлуатаційне призначення розробки полягає в практичному використанні отриманих знань для розвитку та вдосконалення проектів, які використовують документно-орієнтовані бази даних, та впровадженні рекомендацій для вибору оптимального рішення у майбутніх проектах.

44165850.1351-01

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Онсовні вимоги до функцыональних характеристик:

- зчитування та запис даних – дослідження повинно охоплювати здатність обох баз даних до зчитування та запису даних, включаючи операції вставки, оновлення та видалення записів;
- запити та фільтрація – мають бути досліджені можливості складних запитів та фільтрації даних, включаючи підтримку операцій порівняння, логічних умов та інших фільтраційних операцій;
- вкладеність даних – потрібно дослідити здатність баз даних обробляти вкладені дані та робити запити до вкладених полів.
- масштабованість – вимоги до масштабованості повинні включати оцінку швидкості та продуктивності при роботі з великими обсягами даних та при великій кількості одночасних користувачів;
- транзакції та атомарність – дослідження має охоплювати підтримку транзакцій та гарантування атомарності операцій;
- робота з індексами: важливо дослідити можливості створення та використання індексів для оптимізації швидкодії даних та запитів.
- захист даних – має бути досліджена система захисту даних, включаючи доступ до даних та автентифікацію користувачів;
- забезпечення резервного копіювання та відновлення – вимоги до забезпечення можливості створення резервних копій даних та відновлення їх в разі втрати чи пошкодження;
- міграція даних – дослідження повинно включати можливість міграції даних між різними версіями баз даних та іншими системами.
- підтримка реплікації та кластеризації – важливо дослідити можливості створення реплікацій та кластерів для забезпечення високої доступності та надійності даних;

- інтеграція з іншими технологіями – дослідження повинно включати можливість інтеграції з іншими технологіями та сервісами, такими як веб-сервери, мови програмування та інші;
- адміністрування та моніторинг – важливо дослідити інструменти для адміністрування та моніторингу баз даних, включаючи можливості відстеження стану системи та виявлення проблем;
- підтримка текстового та геопросторового пошуку – мають бути досліджені можливості для текстового та геопросторового пошуку даних.

Ці вимоги до функціональних характеристик дослідження документноорієнтованих баз даних MongoDB та Google Firestore допоможуть зрозуміти їхні можливості та вибрати найкращу платформу для конкретних потреб проекту.

Вхідні дані:

- дані про структуру документів – опис структури документів, які будуть зберігатися в базах даних, включаючи назви полів, типи даних та вкладені структури;
- обсяг даних – кількість та об'єм даних, які планується зберігати в базах даних, включаючи кількість записів та обсяг текстової і бінарної інформації;
- методи доступу до даних – опис методів доступу до даних, включаючи створення, зчитування, оновлення та видалення документів;
- сценарії використання – визначення конкретних сценаріїв використання, таких як додавання нових даних, пошук за певними умовами, оновлення та видалення записів;
- серверні налаштування – інформація про конфігурацію серверів, на яких розгорнуті бази даних, включаючи обсяг доступної пам'яті, кількість ядер процесора та інші параметри.

Вихідні дані:

- результати дослідження функціональності – звіт або документ, який містить результати дослідження функціональних можливостей обох баз даних, включаючи порівняльний аналіз їхніх можливостей;
- рекомендації для вибору бази даних – інформація про те, яка з баз даних, MongoDB або Google Firestore, найкраще відповідає вимогам та потребам проекту, з вказівкою на переваги та недоліки кожної з них;
- план впровадження – надається план впровадження ДОБД, включаючи методи міграції даних та інші необхідні кроки.

3.2 Вимоги до надійності

Вимоги до надійності наступні:

- при оновленні даних забезпечити показ відповідних повідомлень про стан роботи програми та результати виконання операцій;
- наявність архівної копії тексту програми на зовнішньому носії;
- наявність резервної копії бази даних на зовнішньому носії.

3.3 Вимоги експлуатації

Працювати з програмою може людина, що має навички роботи з десктопними пристроями та ознайомлена з керівництвом користувача програмного продукту.

3.4 Вимоги до складу та параметрів технічних засобів

Продукти, що розробляється повинен використовуватись на пристроях, що мають наступні характеристики:

- операційна система – комп'ютер повинен працювати під управлінням операційної системи, яка підтримує виконання програмного продукту. Для прикладу, це може бути операційна система Windows, macOS або Linux;
- діагональ монітора – монітор комп'ютера повинен мати достатньою діагональ для зручного відображення інтерфейсу продукту та взаємодії з

ним. Ідеально, це може бути монітор із середньою діагоналлю, наприклад, 21-24 дюйми;

- роздільна здатність монітора – монітор повинен мати достатньо високу роздільну здатність для чіткого та якісного відображення інтерфейсу продукту. Рекомендована роздільна здатність - Full HD (1920x1080) або вища;
- оперативна пам'ять (RAM) – комп'ютер повинен мати достатньо оперативної пам'яті для ефективної роботи програмного продукту. Рекомендована кількість оперативної пам'яті - не менше 4 ГБ;
- вбудована пам'ять (жорсткий диск або SSD) – комп'ютер повинен мати достатньо внутрішньої пам'яті для зберігання програмного продукту та його даних. Рекомендована ємність вбудованої пам'яті - не менше 256 ГБ;
- частота процесора – комп'ютер повинен мати процесор з достатньою частотою для швидкої обробки даних та виконання завдань програмного продукту. Рекомендована частота процесора - не менше 2.5 ГГц;
- порти та комунікації – комп'ютер повинен мати необхідні порти та комунікаційні можливості для підключення до мережі, зовнішніх пристроїв та інтернету. Доцільно мати USB-порти, Ethernet-порт для мережевого підключення, а також можливість підключення до Wi-Fi.

3.5 Вимоги до інформаційної та програмної сумісності

Програмні продукти розробляється для всіх видів операційних систем сімейства “Windows” починаючи від версії 10 та наступні версії.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

– текст програми;

– керівництво користувача для користування мобільним додатком.

Вся документація програмних додатків повинна задовольняти вимоги до програмної документації.

44165850.1351-01

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця 5.1 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	04.09.23 –15.09.23
Робочий проект	Програмування та відлагодження програми.	18.09.23 – 22.09.23
	Тестування програми	25.09.23 – 27.09.23
	Розробка, узгодження і затвердження програмної документації.	28.09.23 – 29.09.23

6 ПОРЯДОК ПРИЙНЯТТЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Іванов О. П.

7 ТЕХНІЧНО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Показники та їх розрахунок не були описані у документах бо розробка програмного продукту несе в собі навчальний характер, а не комерційний.

ДОДАТОК Б

Технічне завдання

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«ДОСЛІДЖЕННЯ ФУНКІОНАЛЬНИХ МОЖЛИВОСТЕЙ ДОКУМЕНТНООРІЄНТОВАНИХ БАЗ ДАНИХ MONGODB ТА GOOGLE FIRESTORE»
Текст програми
44165850.01223–01 12–01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Максим ПЛЯШКО
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО

44165850.01223–01 12–01

ДОСЛІДЖЕННЯ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ ДОКУМЕНТ-ОРІЄНТОВАНИХ БАЗ ДАНИХ MONGODB ТА GOOGLE FIRESTORE

Текст програми
44165850.01223–01 12–01
Листів 17

2024

АНОТАЦІЯ

Документ 44165850.01223–01 12 01 «Дослідження функціональних можливостей документ-орієнтованих баз даних MongoDB та Google Firestore» входить до складу програмної документації на програму, що виконує роль прототипу додатку математичного задачника для навчання дітей арифметиці.

У даному документі представлений текст додатку. Додаток написаний на мові C# з використанням бібліотек Google.Cloud.Firestore та MongoDB.Driver у програмному середовищі Visual Studio 2022.

ЗМІСТ

44165850.01223–01 12

1 ТЕКСТ ПРОГРАМИ

1.1 Текст контроллера FirestoreController.cs

```
using Amazon.Auth.AccessControlPolicy;
using Google.Cloud.Firestore;
using MongoDB.Driver;
using NLog;
using System.Diagnostics;
using WinFormsApp1;
using static
Google.Cloud.Firestore.V1.StructuredAggregationQuery.Types.Aggr
egation.Types;

namespace ConsoleApp2.Google
{
   internal class FirestoreController
   {
      private readonly FirestoreDb _db;
      private CollectionReference _collectionReference;
      Random Random = new Random();
      string projectName = "test-1de66";
      private static Logger logger =
LogManager.GetCurrentClassLogger();

      public FirestoreController()
      {
         string keyFilePath = Path.Combine("../../../firestore.json");

Environment.SetEnvironmentVariable("GOOGLE_APPLICATION_
CREDENTIALS", keyFilePath);

         _db = FirestoreDb.Create(projectName);

         try
         {
            _collectionReference = _db.Collection("qwe");
         }
         catch (Exception ex)
         {
            Console.WriteLine($"Error creating collection:
{ex.Message}");
         }
      }

      private async Task AddDataToCollection(Dictionary<string,
object> keyValues)
      {

         DocumentReference docRef =
_collectionReference.Document();
         docRef.SetAsync(keyValues).Wait();
      }
      public async Task
AddToDB(List<WinFormsApp1.Interfaces.Pair<string, string[]>>
pairList, int count)
      {
         _collectionReference = _db.Collection("qwe");
         logger.Info("Add data to data base");

         Stopwatch stopwatch = new Stopwatch();
         using (var customMessageBox = new MyMessageBox())
         {
            long elapsedTimeMs = 0;
            int countOfAdding = 0;


            customMessageBox.Show();
            for (int i = 0; i < count; ++i)
            {
               customMessageBox.UpdateMessage($"{i + 1}");

               Dictionary<string, object> data =
ConvertToDictionary(pairList);
               stopwatch.Start();
               await AddDataToCollection(data);
               stopwatch.Stop();
               elapsedTimeMs += stopwatch.ElapsedMilliseconds;
               stopwatch.Reset();
               countOfAdding = i;
            }

            logger.Info($"Adding {countOfAdding + 1} times of data
spend {elapsedTimeMs}ms");
         }
      }
      private Dictionary<string, object>
ConvertToDictionary(List<WinFormsApp1.Interfaces.Pair<string,
string[]>> keyValues)
      {
         Dictionary<string, object> result = new Dictionary<string,
object>();

         foreach (var pair in keyValues)
         {
            int count = Random.Next(0, pair.Second.Length);
            result[pair.First] = pair.Second[count];
         }

         return result;
      }


      public async Task<List<List<string>>>
GetData(List<List<string>> keys)
      {
         var q = _db.Collection("qwe");
         _collectionReference = q;
         var condition = CreateQuery(keys);

         logger.Info("Get data from data base");

         Stopwatch stopwatch = new Stopwatch();
         var a = new List<DocumentSnapshot>();
         long elapsedTimeMs = 0;

         bool flag = true;

         _ = Task.Run(async () =>
         {
            using (var customMessageBox = new MyMessageBox())
            {
               customMessageBox.Show();
               customMessageBox.UpdateMessage($"GetData");
               while (flag) ;
            }
         });


         stopwatch.Start();
         a = await GetDataWithCondition(condition,
keys).ConfigureAwait(false);
         stopwatch.Stop();
         elapsedTimeMs += stopwatch.ElapsedMilliseconds;
```

```
        stopwatch.Reset();
        flag = false;

        logger.Info($"Getting data spend {elapsedTimeMs}ms");

        List<List<string>> dataList = new List<List<string>>();

        foreach (var documentSnapshot in a)
        {
            List<string> documentData = new List<string>();
            foreach (var field in documentSnapshot.ToDictionary())
            {
                documentData.Add($"{field.Key}: {field.Value}");
            }
            documentData.Reverse();
            dataList.Add(documentData);
        }

        _collectionReference = null;
        return dataList;

    }
    public async Task<List<DocumentSnapshot>>
GetDataWithCondition(Dictionary<string, Query> conditions,
List<List<string>> keys)
    {
        var documentDictionary = new Dictionary<string,
List<DocumentSnapshot>>();
        if (conditions.Count == 0 )
        {
            QuerySnapshot querySnapshot = await
_collectionReference.GetSnapshotAsync().ConfigureAwait(false); ;
            return querySnapshot.Documents.ToList();
        }

        foreach (var condition in conditions)
        {
            var snapshot = await
condition.Value.GetSnapshotAsync().ConfigureAwait(false);
            return snapshot.Documents.ToList();
        }

        int count = -1;
        while (keys.Count != 0)
        {
            count++;
            var key = keys[count];

            if (documentDictionary.ContainsKey(key[0]) &&
documentDictionary.ContainsKey(key[2]))
            {
                var r = new List<DocumentSnapshot>();
                switch (key[1])
                {
                    case "AND": r =
CombineSnapshotsWithAnd(documentDictionary[key[0]],
documentDictionary[key[2]]); break;
                    case "OR": r =
CombineSnapshotsWithOr(documentDictionary[key[0]],
documentDictionary[key[2]]); break;
                }
                documentDictionary.Add(key[3], r);
                keys.RemoveAt(count);
                count = -1;
                if (keys.Count == 0)
                {
                    return r;
                }
            }
        }
        var a = new List<DocumentSnapshot>();
        return a;

    }

    public List<DocumentSnapshot>
CombineSnapshotsWithAnd(List<DocumentSnapshot> snapshots1,
List<DocumentSnapshot> snapshots2)
    {
        var documentIds1 = snapshots1.Select(doc =>
doc.Id).ToList();
        var commonDocuments = snapshots2.Where(doc =>
documentIds1.Contains(doc.Id)).ToList();

        return commonDocuments;
    }

    public List<DocumentSnapshot>
CombineSnapshotsWithOr(List<DocumentSnapshot> snapshots1,
List<DocumentSnapshot> snapshots2)
    {
        var combinedDocuments =
snapshots1.Concat(snapshots2).ToList();

        return combinedDocuments;
    }

    public Dictionary<string, Query>
CreateQuery(List<List<string>> keys)
    {
        if (keys == null || keys.Count == 0)
        {
            Dictionary<string, Query> a = new Dictionary<string,
Query>();
            return a;
        }

        Dictionary<string, Query> queries = new Dictionary<string,
Query>();

        foreach (var key in keys.ToList())
        {
            if (key[1] != "AND" && key[1] != "OR")
            {
                Query query = GetSimpleQuery(key);

                if (query != null)
                {
                    keys.Remove(key);
                    queries.Add(key[3], query);
                }
            }
        }

        return queries;
    }
    private Query GetSimpleQuery(List<string> key)
    {
        string field = key[0];
        string comparisonOperator = key[1];
        string value = key[2];

        switch (comparisonOperator)
        {
            case "<":
                return _collectionReference.WhereLessThan(field,
value);
            case "<=":
                return
_collectionReference.WhereLessThanOrEqualTo(field, value);
            case ">":
                return _collectionReference.WhereGreaterThan(field,
value);
            case ">=":
                return
_collectionReference.WhereGreaterThanOrEqualTo(field, value);
            case "=":
                return _collectionReference.WhereEqualTo(field,
value);
```

```
            case "!=":
                return _collectionReference.WhereNotEqualTo(field,
value);
            default:
                return null;
        }
    }

    public async Task DeleteData(List<List<string>> keys)
    {
        _collectionReference = _db.Collection("qwe");
        logger.Info("Get data from data base");

        Stopwatch stopwatch = new Stopwatch();
        long elapsedTimeMs = 0;
        bool flag = true;

        _ = Task.Run(async () =>
        {
            using (var customMessageBox = new MyMessageBox())
            {
                customMessageBox.Show();
                customMessageBox.UpdateMessage($"Delete Data");
                while (flag) ;
            }
        });

        stopwatch.Start();
        await DeleteDataFromDB(keys).ConfigureAwait(false);
        stopwatch.Stop();
        elapsedTimeMs += stopwatch.ElapsedMilliseconds;
        stopwatch.Reset();
        flag = false;
        logger.Info($"Deleting data spend {elapsedTimeMs}ms");

    }

    public async Task DeleteDataFromDB(List<List<string>>
keys)
    {
        List<Query> queries = new List<Query>();

        foreach (var key in keys)
        {
            queries.Add(GetSimpleQuery(key));
        }

        try
        {
            if (queries.Count == 0)
            {

                QuerySnapshot querySnapshot = await
_collectionReference.GetSnapshotAsync().ConfigureAwait(false);

                foreach (DocumentSnapshot documentSnapshot in
querySnapshot.Documents)
                {
                    DocumentReference documentReference =
documentSnapshot.Reference;
                    await documentReference.DeleteAsync();
                }

            }
            else
            {
                Query query = queries.First();
                QuerySnapshot querySnapshot = await
query.GetSnapshotAsync().ConfigureAwait(false);

                foreach (DocumentSnapshot documentSnapshot in
querySnapshot.Documents)
                {
                    DocumentReference documentReference =
documentSnapshot.Reference;
                    documentReference.DeleteAsync();

                }
            }
        }
        catch (Exception ex)
        {
            logger.Error($"Error deleting documents in collection:
{ex.Message}");
        }
    }

    public async Task UpdateDataInDB(List<List<string>> keys,
List<List<string>> update)
    {
        _collectionReference = _db.Collection("qwe");
        Dictionary<string, object> dictionary = new
Dictionary<string, object>();

        foreach (var item in update)
        {
            if (item.Count >= 2)
            {
                string key = item[0];
                string value = item[1];

                if (dictionary.ContainsKey(key))
                {
                    dictionary[key] = value;
                }
                else
                {
                    dictionary.Add(key, value);
                }
            }
        }

        long elapsedTimeMs = 0;
        using (var customMessageBox = new MyMessageBox())
        {

            customMessageBox.Show();

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            UpdateAllDocumentsWithCondition(keys, dictionary);
            stopwatch.Stop();
            elapsedTimeMs += stopwatch.ElapsedMilliseconds;
            stopwatch.Reset();

        }

        logger.Info($"Update data spend {elapsedTimeMs}ms");
    }

    public async Task
UpdateAllDocumentsWithCondition(List<List<string>> keys,
Dictionary<string, object> newData)
    {
        List<Query> queries = new List<Query>();
        if (keys.Count == 0)
        {
            foreach (var key in keys)
            {
                queries.Add(GetSimpleQuery(key));
            }
        }

        try
        {
            QuerySnapshot querySnapshot;
            if (queries.Count != 0)
            {
                Query query = queries.First();
                querySnapshot = await query.GetSnapshotAsync();
```

```
            }
            else
            {
                querySnapshot = await
_collectionReference.GetSnapshotAsync();
            }

            foreach (DocumentSnapshot documentSnapshot in
querySnapshot.Documents)
            {
                DocumentReference documentReference =
documentSnapshot.Reference;
                await documentReference.UpdateAsync(newData);
                Console.WriteLine($"Document updated in collection
'{_collectionReference.Id}' with ID: {documentReference.Id}");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error updating documents in
collection: {ex.Message}");
        }
    }
  }
}
```

1.2 Текст контроллера mongoDBController.cs

```
using MongoDB.Bson;

using MongoDB.Driver;

using NLog;

using System.Diagnostics;

using System.Windows.Forms;

using WinFormsApp1;

using static
Google.Cloud.Firestore.V1.StructuredAggregationQuery.Types.Aggr
egation.Types;


namespace MongoDB.mongoDBController

{

  internal class MongoDBController

  {

    private static Logger logger =
LogManager.GetCurrentClassLogger();


    private readonly string databaseName = "testDB";

    private readonly string collectionName = "123";

    private MongoClient _client => new
MongoClient("mongodb+srv://max467364:maxsys2001@cluster0.zj
pwsib.mongodb.net/");



    private IMongoCollection<BsonDocument> _collection;

    private BsonDocument structure;


    Dictionary<string, int> operatorPriority = new
Dictionary<string, int>

    {

      {"<", 0 },

      {"<=", 1 },

      {">", 2 },

      {">=", 3 },

      {"=", 4 },

      {"!=", 5 },

      {"AND", 6 },

      {"OR", 7 },

      {"", 8 },

      {"ANY", 9 }

    };



    public void SetCollection()

    {

      logger.Info("Set collection");

      _collection =
_client.GetDatabase(databaseName).GetCollection<BsonDocument>
(databaseName);

    }


    public void SetDocumentStructure(List<string> keys)

    {
```

```
        logger.Info("Set structure ", keys);
        structure = new BsonDocument(keys.ToDictionary(key => key, _ => ""));
    }

    public bool chechFilterIsUse(List<List<string>> keys, string filterid)
    {
        foreach (var key in keys.ToList())
        {
            if (key[0] == filterid || key[2] == filterid)
                return true;
        }
        return false;
    }
    public FilterDefinition<BsonDocument> CreateFilter(List<List<string>> keys)
    {
        if (keys == null || keys.Count == 0)
        {
            return Builders<BsonDocument>.Filter.Empty;
        }

        var result = new List<FilterDefinition<BsonDocument>>();
        var simplefilters = new Dictionary<string, FilterDefinition<BsonDocument>>();

        foreach (var key in keys.ToList())
        {
            if (key[1] != "AND" || key[1] != "OR")
            {
                FilterDefinition<BsonDocument> filter = GetSimpleFilter(key);

                if (filter != null)
                {
                    keys.Remove(key);
                    simplefilters.Add(key[3], filter);
                }
            }
        }

        foreach (var sf in simplefilters)
        {
            if (!chechFilterIsUse(keys, sf.Key))
            {
                result.Add(sf.Value);
            }
        }

        var filters = simplefilters;

        while (keys.Count > 0)
        {

            foreach (var key in keys.ToList())
            {
                if (filters.ContainsKey(key[0]) && filters.ContainsKey(key[2]))
                {
                    FilterDefinition<BsonDocument> advancedFilter = GetAdvancedFilter(key, filters[key[0]], filters[key[2]]);

                    var key1 = key[0];
                    var key2 = key[2];
                    if (advancedFilter != null)
                    {
                        keys.Remove(key);
                        filters.Add(key[3], advancedFilter);
                    }

                    if (!chechFilterIsUse(keys, key1))
                    {
```

```
                filters.Remove(key1);
            }

            if (!chechFilterIsUse(keys, key2))
            {
                filters.Remove(key2);
            }
        }
    }

    result.AddRange(filters.Values);
    return Builders<BsonDocument>.Filter.Or(result);
}

private FilterDefinition<BsonDocument>
GetSimpleFilter(List<string> key)
{
    string field = key[0];
    string comparisonOperator = key[1];
    string value = key[2];

    if (!operatorPriority.TryGetValue(comparisonOperator, out
int priority))
    {
        return null;
    }

    switch (priority)
    {
        case 0:
            return Builders<BsonDocument>.Filter.Lt(field, value);
        case 1:
            return Builders<BsonDocument>.Filter.Lte(field, value);
        case 2:
            return Builders<BsonDocument>.Filter.Gt(field, value);
        case 3:
            return Builders<BsonDocument>.Filter.Gte(field,
value);
        case 4:
            return Builders<BsonDocument>.Filter.Eq(field, value);
        case 5:
            return Builders<BsonDocument>.Filter.Ne(field, value);
        case 8:
            return Builders<BsonDocument>.Filter.Empty;
        case 9:
            return Builders<BsonDocument>.Filter.Exists(field);
        default:
            return null;
    }
}
private FilterDefinition<BsonDocument>
GetAdvancedFilter(List<string> key,
FilterDefinition<BsonDocument> filter1,
FilterDefinition<BsonDocument> filter2)
{
    string comparisonOperator = key[1];

    if (!operatorPriority.TryGetValue(comparisonOperator, out
int priority))
    {
        return null;
    }

    switch (priority)
    {
        case 6:
            return Builders<BsonDocument>.Filter.And(filter1,
filter2);
        case 7:
            return Builders<BsonDocument>.Filter.Or(filter1,
filter2);
```

```
            default:
                return null;
        }
    }

    public async Task AddToDB(int count,
List<WinFormsApp1.Interfaces.Pair<string, string[]>> keyValues)
    {

        logger.Info("Add data to data base");
        if (!CollectionExists())
        {

_client.GetDatabase(databaseName).CreateCollection(collectionNam
e);
        }

         _collection =
_client.GetDatabase(databaseName).GetCollection<BsonDocument>
(collectionName);

        long elapsedTimeMs = 0;
        int countOfAdding = 0;
        Stopwatch stopwatch = new Stopwatch();

        using (var customMessageBox = new MyMessageBox())
        {
            customMessageBox.Show();

            try
            {
                for (int i = 0; i < count; ++i)
                {
                    customMessageBox.UpdateMessage($"{i + 1}");

                    var variationDocument = new
BsonDocument(structure);
                    foreach (var keyValue in keyValues)
                        variationDocument[keyValue.First] =
GetRandomValue(keyValue.Second);

                    stopwatch.Start();

                    _collection.InsertOne(variationDocument);

                    stopwatch.Stop();

                    elapsedTimeMs += stopwatch.ElapsedMilliseconds;
                    stopwatch.Reset();
                    countOfAdding = i;
                }
            }
            catch(Exception ex)
            {
                logger.Info($"Error to add data to DB");
                logger.Info(ex);

            }
        }
        logger.Info($"Adding {countOfAdding + 1} times of data
spend {elapsedTimeMs}ms");

    }

    private static T GetRandomValue<T>(T[] array)
    {
        if (array == null || array.Length == 0)
        {
            throw new ArgumentException("The array is null or
empty.");
        }
```

```
        Random random = new Random();
        int randomIndex = random.Next(0, array.Length);

        return array[randomIndex];
    }

    private List<List<string>>
ConvertBsonDocumentsToStringList(List<BsonDocument>
bsonDocuments)
    {
        List<List<string>> result = new List<List<string>>();
        foreach (BsonDocument bsonDocument in bsonDocuments)
        {
            List<string> documentStrings = new List<string>();

            foreach (BsonElement element in
bsonDocument.Elements)
            {
                string fieldName = element.Name;
                string fieldValue = element.Value.ToString();
                documentStrings.Add($"{fieldName}: {fieldValue}");
            }
            result.Add(documentStrings);
        }

        return result;
    }
    public List<List<string>> GetData(List<List<string>> keys)
    {
        var filter = CreateFilter(keys);

        Stopwatch stopwatch = new Stopwatch();
        using (var customMessageBox = new MyMessageBox())
        {
            customMessageBox.UpdateMessage($"Get data");
            stopwatch.Start();
            var data = GetDataWithConditionDB(filter);
            stopwatch.Stop();

            long elapsedTimeMs = stopwatch.ElapsedMilliseconds;
            stopwatch.Reset();
            logger.Info($"Getting data spend {elapsedTimeMs}ms");
            return ConvertBsonDocumentsToStringList(data);
        }

    }
    public UpdateDefinition<BsonDocument>
GetUpdate(List<List<string>> keys)
    {
        var updateDefinitions = new
List<UpdateDefinition<BsonDocument>>();

        foreach (var key in keys)

updateDefinitions.Add(Builders<BsonDocument>.Update.Set(key[0]
, key[1]));

        var combinedUpdate =
Builders<BsonDocument>.Update.Combine(updateDefinitions);
        return combinedUpdate;

    }

    public void UpdateDataInDB(List<List<string>> filterData,
List<List<string>> updateData)
    {
      var filter = CreateFilter(filterData);

        Stopwatch stopwatch = new Stopwatch();
        using (var customMessageBox = new MyMessageBox())
        {
            customMessageBox.Show();
```

```
            customMessageBox.UpdateMessage($"Update data");

            stopwatch.Start();

            var update = GetUpdate(updateData);
            UpdateWithConditionDB(filter, update);
            stopwatch.Stop();

            long elapsedTimeMs = stopwatch.ElapsedMilliseconds;
            stopwatch.Reset();

            logger.Info($"Update data spend {elapsedTimeMs}ms");
        }
    }

    public void
UpdateWithConditionDB(FilterDefinition<BsonDocument> filter,
UpdateDefinition<BsonDocument> update)
    {
        _collection =
_client.GetDatabase(databaseName).GetCollection<BsonDocument>
(collectionName);

        _collection.UpdateMany(filter, update);
    }

    public void DeleteData(List<List<string>> keys)
    {

        var filter = CreateFilter(keys);

        Stopwatch stopwatch = new Stopwatch();
        using (var customMessageBox = new MyMessageBox())
        {
            customMessageBox.UpdateMessage($"Delete data");
            stopwatch.Start();

            DeleteWithConditionDB(filter);
            stopwatch.Stop();

            long elapsedTimeMs = stopwatch.ElapsedMilliseconds;
            stopwatch.Reset();

            logger.Info($"Deleting data spend {elapsedTimeMs}ms");

        }
    }
    public void
DeleteWithConditionDB(FilterDefinition<BsonDocument> filter)
    {
        _collection =
_client.GetDatabase(databaseName).GetCollection<BsonDocument>
(collectionName);

        _collection.DeleteMany(filter);
    }

    public List<BsonDocument>
GetDataWithConditionDB(FilterDefinition<BsonDocument> filter)
    {
        _collection =
_client.GetDatabase(databaseName).GetCollection<BsonDocument>
(collectionName);
        ProjectionDefinition<BsonDocument> projection =
Builders<BsonDocument>.Projection.Exclude("_id");

        var documents =
_collection.Find(filter).Project(projection).ToList();

        return documents;
    }

    private bool CollectionExists()
    {
        var filter = new BsonDocument("name", collectionName);
```

```
        var collections =
_client.GetDatabase(databaseName).ListCollections(new
ListCollectionsOptions { Filter = filter });

        return collections.Any();
```

```
      }

    }

}
```

1.3 Текст файлу Form1.cs

```
using MongoDB.mongoDBController;
using ConsoleApp2.Google;
using WinFormsApp1.Interfaces;
using NLog;

namespace WinFormsApp1
{
  public partial class Form1 : Form
  {
    MongoDBController mb = new MongoDBController();
    FirestoreController fr = new FirestoreController();
    private int selectedIndexLB1 = -1;
    enum BD
    {
      MONODB = 0,
      FIRESTORE = 1,
      NODB
    }
    BD currentBase = BD.NODB;
    public Form1()
    {
      InitializeComponent();
      KeyList.SelectionMode = SelectionMode.MultiSimple;
    }

    private void Form1_Load(object sender, EventArgs e)
    {
      mb.SetCollection();
      AddColumn("Key", "Key 1", 30);
      AddColumn("PosibleValue", "PosibleValue", 46);

      conditionsGridView.Columns.Add("Key1", "Key1");

      DataGridViewComboBoxColumn comboBoxColumn = new
DataGridViewComboBoxColumn();
      comboBoxColumn.Name = "Operation";
      comboBoxColumn.HeaderText = "Operation";
      comboBoxColumn.Items.Add("<");
      comboBoxColumn.Items.Add("<=");
      comboBoxColumn.Items.Add(">");
      comboBoxColumn.Items.Add(">=");
      comboBoxColumn.Items.Add("=");
      comboBoxColumn.Items.Add("!=");
      comboBoxColumn.Items.Add("AND");
      comboBoxColumn.Items.Add("OR");
      comboBoxColumn.Items.Add("ANY");

      conditionsGridView.Columns.Add(comboBoxColumn);

      conditionsGridView.Columns.Add("Key2", "Key2");
      conditionsGridView.Columns.Add("Name of operation",
"Name");

      updateValueGridView.Columns.Add("Key", "Key 1");
      updateValueGridView.Columns.Add("Value", "Value");

      keyValueGridView.CellDoubleClick +=
DataGridView_CellDoubleClick;
    }

    private void ModifySelectedItems()
    {
      if (KeyList.SelectedItem != null)
      {

        string modifiedItem =
ModifyItem(KeyList.SelectedItem.ToString());
        if (!string.IsNullOrWhiteSpace(modifiedItem))
        {
          KeyList.Items[KeyList.SelectedIndex] = modifiedItem;
        }
```

```
        }

        KeyList.SelectedItem = null;
    }

    private void AddNewItem()
    {
        string newValue = Microsoft.VisualBasic.Interaction.InputBox("Enter the new value:", "Add Item");
        if (!string.IsNullOrWhiteSpace(newValue))
        {
            KeyList.Items.Add(newValue);
        }
    }

    private void RemoveSelectedItems()
    {
        if (KeyList.SelectedIndex != -1)
        {
            KeyList.Items.RemoveAt(selectedIndexLB1);
            KeyList.SelectedIndex = -1;
        }
    }

    private string ModifyItem(string item)
    {
        return Microsoft.VisualBasic.Interaction.InputBox("Enter the new value:", "Modify Item", item);
    }
    private void DataGridView_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
    {
        var datagrid = (DataGridView)sender;
        string newValue = Microsoft.VisualBasic.Interaction.InputBox(
                        "Old value: " +
datagrid.Rows[e.RowIndex].Cells[e.ColumnIndex].Value +
                        "\nEnter new value:",
                        "Add new value");

        if (!string.IsNullOrWhiteSpace(newValue))
        {
            datagrid.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = newValue;
        }
    }

    private void AddColumn(string name, string headerText, int widthPercentage)
    {
        DataGridViewTextBoxColumn column = new DataGridViewTextBoxColumn();
        column.Name = name;
        column.HeaderText = headerText;
        column.Width = (int)Math.Round(keyValueGridView.Width * widthPercentage / 100.0);
        keyValueGridView.Columns.Add(column);
    }

    private List<List<string>> GetDataFromGrid(DataGridView datagrid)
    {
        List<List<string>> data = new List<List<string>>();
        foreach (DataGridViewRow row in datagrid.Rows)
        {
            if (!row.IsNewRow)
            {
                List<string> rowData = new List<string>();
                foreach (DataGridViewCell cell in row.Cells)
                {
                    rowData.Add(cell.Value?.ToString() ?? string.Empty);
                }
                data.Add(rowData);
            }
        }
        return data;
    }

    private void ClearUpdateData_Click(object sender, EventArgs e)
    {
        updateValueGridView.Rows.Clear();
    }

    private void KeyList_DoubleClick(object sender, EventArgs e)
    {
        ModifySelectedItems();
    }

    private void KeyList_Click(object sender, EventArgs e)
    {
        selectedIndexLB1 = KeyList.SelectedIndex;
    }

    private void AddKeyButton_Click(object sender, EventArgs e)
```

```
        {
            AddNewItem();
        }

        private void RemoveKeyButton_Click(object sender, EventArgs
e)
        {
            RemoveSelectedItems();
        }

        private void mongoDBRadioButton_CheckedChanged(object
sender, EventArgs e)
        {
            currentBase = BD.MONODB;
        }

        private void fireStoreradioButton_CheckedChanged(object
sender, EventArgs e)
        {
            currentBase = BD.FIRESTORE;
        }

        private void SetStructureButton_Click(object sender, EventArgs
e)
        {
            keyValueGridView.Rows.Clear();
            int count = KeyList.Items.Count;
            for (int i = 0; i < count - 1; ++i)
            {
                for (int j = i + 1; j < count - 1; ++j)
                {
                    if (KeyList.Items[i].ToString() ==
KeyList.Items[j].ToString())
                    {
                        KeyList.Items.RemoveAt(j);
                        --j;
                        --count;
                    }
                }
            }

            List<string> keyList = new List<string>();
            foreach (object item in KeyList.Items)
            {
                keyValueGridView.Rows.Add(item.ToString(), "");
                keyList.Add(item.ToString());
            }
            mb.SetDocumentStructure(keyList);
        }

        private void AddDataButton_Click(object sender, EventArgs e)
        {
            List<WinFormsApp1.Interfaces.Pair<string, string[]>>
pairList = new List<WinFormsApp1.Interfaces.Pair<string,
string[]>>();

            foreach (DataGridViewRow row in keyValueGridView.Rows)
            {
                if (row.Cells["PosibleValue"].Value != null)
                {
                    string dataFromCell =
row.Cells["PosibleValue"].Value.ToString();
                    Pair<string, string[]> a = new
WinFormsApp1.Interfaces.Pair<string,
string[]>(row.Cells["Key"].Value.ToString(),
dataFromCell.Split(','));
                    pairList.Add(a);
                }
            }
            int.TryParse(countOfInserting.Text, out int count);

            switch (currentBase)
            {
                case BD.MONODB: mb.AddToDB(count, pairList); break;
                case BD.FIRESTORE: fr.AddToDB(pairList,
count).Wait(); break;
                case BD.NODB: { MessageBox.Show("Database didn't
choose", "Alert", MessageBoxButtons.OK,
MessageBoxIcon.Information); return; }
            }
        }

        private void GetDataButton_Click(object sender, EventArgs e)
        {
            var filterData = new List<List<string>>();
            filterData = GetDataFromGrid(conditionsGridView);
            List<List<string>> data = new List<List<string>>();

            switch (currentBase)
            {
                case BD.NODB: { MessageBox.Show("Database didn't
choose", "Alert", MessageBoxButtons.OK,
MessageBoxIcon.Information); return; }
                case BD.MONODB: data = mb.GetData(filterData); break;
                case BD.FIRESTORE:
                    {
                        var task = fr.GetData(filterData);
```

```
                task.Wait();
                data = task.Result;
                break;
            }
        }

        outputListBox.Items.Clear();
        foreach (var dt in data)
        {
            string itemString = "{";
            foreach (var d in dt)
            {
                itemString += d + " ";
            }
            itemString += "}";
            outputListBox.Items.Add(itemString);
        }
        outputListBox.Items.Add($"Items : {outputListBox.Items.Count.ToString()}");
    }

    private void DeleteDataButton_Click(object sender, EventArgs e)
    {

        var filterData = new List<List<string>>();

        foreach (DataGridViewRow row in conditionsGridView.Rows)
        {
            if (!row.IsNewRow)
            {
                List<string> rowData = new List<string>();

                foreach (DataGridViewCell cell in row.Cells)
                {
                    rowData.Add(cell.Value?.ToString() ?? string.Empty);
                }

                filterData.Add(rowData);
            }
        }

        switch (currentBase)
        {
            case BD.MONODB: { mb.DeleteData(filterData); break; }
            case BD.NODB: { MessageBox.Show("Database didn't choose", "Alert", MessageBoxButtons.OK, MessageBoxIcon.Information); break; }
            case BD.FIRESTORE:
                {
                    var task = fr.DeleteData(filterData);
                    task.Wait();
                    break;
                }
        }
    }

    private void ClearConditions_Click(object sender, EventArgs e)
    {
        conditionsGridView.Rows.Clear();
    }

    private void UpdateDataButton_Click(object sender, EventArgs e)
    {
        List<List<string>> filterData = new List<List<string>>();
        List<List<string>> updateData = new List<List<string>>();

        foreach (DataGridViewRow row in conditionsGridView.Rows)
        {
            if (!row.IsNewRow)
            {
                List<string> rowData = new List<string>();

                foreach (DataGridViewCell cell in row.Cells)
                {
                    rowData.Add(cell.Value?.ToString() ?? string.Empty);
                }

                filterData.Add(rowData);
            }
        }

        foreach (DataGridViewRow row in updateValueGridView.Rows)
        {
            if (!row.IsNewRow)
            {
                List<string> rowData = new List<string>();

                foreach (DataGridViewCell cell in row.Cells)
```

```
            {
                rowData.Add(cell.Value?.ToString() ?? string.Empty);
            }

            updateData.Add(rowData);
        }
    }

    switch (currentBase)
    {
        case BD.MONODB: mb.UpdateDataInDB(filterData, updateData); break;
        case BD.FIRESTORE:
            {
                var task = fr.UpdateDataInDB(filterData, updateData);
                task.Wait();
                break;
            }
        case BD.NODB: { MessageBox.Show("Database didn't choose", "Alert", MessageBoxButtons.OK, MessageBoxIcon.Information); return; }
    }
  }
}
```

ДОДАТОК В

Технічно завдання

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«ДОСЛІДЖЕННЯ ФУНКІОНАЛЬНИХ МОЖЛИВОСТЕЙ ДОКУМЕНТНООРІЄНТОВАНИХ БАЗ ДАНИХ MONGODB ТА GOOGLE FIRESTORE»
Керівництво користувача
44165850.01351 – 01 І3 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Максим ПЛЯШКО
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО
44165850.01351-01 І3 01

«ДОСЛІДЖЕННЯ ФУНКІОНАЛЬНИХ МОЖЛИВОСТЕЙ ДОКУМЕНТНООРІЄНТОВАНИХ БАЗ ДАНИХ MONGODB ТА GOOGLE FIRESTORE»

Керівництво користувача
44165850.01351 – 01 І3 01

Листів 9

2024

44165850.01351-01 І3 01

А Н О Т А Ц І Я

Документ 1116130.01269 – 01 І3 01 «Дослідження функціональних можливостей документ-орієнтованих баз даних MongoDB та Google Firestore. Керівництво користувача».

Програми написані на мові C# з використанням бібліотек Google.Cloud.Firestore та MongoDB.Driver;у програмному середовищі Visual Studio 2022.

44165850.01351-01 І3 01

З М І С Т

1 ВВЕДЕННЯ

Програмний засіб “Дослідження функціональних можливостей документ-орієнтованих баз даних MongoDB та Google Firestore” та призначений для дослідження та аналізу можливостей документ-орієнтованих баз даних.

Головною метою розробки є надання достатнього інструментарію керування роботою з базами даних проведення дослідження та збирання часових характеристик операцій, на основі яких буде проводитися дослідження.

Даний продукт призначений для використання науковцями для дослідження різних способів взаємодії з програмним продуктом та визначення найбільш вдалого вибору базу бази даних для конкретних цілей, але не потребує високого рівня досвіду для використання, та може бути використаний буть-ким.

Не вимагає ознайомлення із будь-якою додатковою експлуатаційною документацією.

2 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Функціональним призначенням додатку є проведення аналізу функціональних можливостей задля вибору більш практичного вибору ДОБД в конкретних умовах.

Експлуатаційне призначення додатку “Дослідження функціональних можливостей документ-орієнтованих баз даних MongoDB та Google Firestore” – робота з базами даних MongoDB та Google Firestore.

Для забезпечення сталого функціонування програми користувачеві і програмісту необхідно дотримуватися таких умов - наявність інтернет з’єднання.

Додаток, що розробляється, розрахований на використання на пристроях що мають:

- процесор з тактовою частотою не нижче 2.5 ГГц;
- не менше 4ГБ оперативної пам’яті.

Для функціонування мобільного додатку неохідна система Windows 10 версії і вище.

3 ПІДГОТОВКА ДО РОБОТИ

Для початку роботи додатку необхідно перемістити .exe файл та конфігуруючий файл для Google Firestore в одну директорію.

4 ОПИС ОПЕРАЦІЙ

Після встановлення та запуску додатку “ ” на формі додатку присутні:

- поле вводу нової структури документу:
- кнопка «+» яка додає нове поле до структури документу:
- кнопка «-» яка видаляє вибране поле з структури документу;
- кнопка «Set Struct» яка зберігає структуру документа та дозволяє вводити дані в таблицю даних;
- кнопка «Add to DB» яка виконує запит додавання документу відповідної структури та даних до бази даних. Кількість документів вводиться в поле біля кнопки;
- таблиця створення фільтру який буде використовуватися для пошуку, видалення та оновлення документів які задовольняють фільтр;
- список результату;
- кнопка «Get data» яка виконує запит отримання документів які задовольняють умовам фільтру, якщо фільтр порожній отримуються всі документи з колекції. Отримані дані будуть відображені в списку результату.
- кнопка «Delete data» яка виконує запит видалення документів які задовольняють умовам фільтру, якщо фільтр порожній видаляться всі документи в колекції;
- кнопка «Clear» яка виконує очищення фільтру;
- таблиця створення нових даних які будуть записані в документ;
- кнопка «Update data» яка виконує запит оновлення документів які задовольняють умовам фільтру, якщо фільтр порожній дані з таблиці нових даних запишуться у всі документи в колекції;
- кнопка «Clear» яка виконує очищення нових даних;
- чек бокс вказує яка БД використовується в поточний момент;

5 АВАРІЙНІ СИТУАЦІЇ

Якщо програмний засіб буде запускатися з пристрою де немає інтернет з’єднання, додаток виведе на екран повідомлення про помилку та завершить роботу.

Якщо програмний засіб під час роботи буде поводити некоректно чи із помилкою, необхідно перезапустити мобільним додаток для продовження роботи.

6 РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

Після встановлення додатку на пристрій його необхідно запустити через виконуючий файл.

Після запуску додатку перед користувачем відкриється головний екран з усім необхідним функціоналом.

Після ознайомлення із функціями додатку, його роботу можна звершити натиснувши апаратну кнопку вимкнення.