Міністерство освіти і науки України

Відокремлений структурний підрозділ

«Зборівський фаховий коледж

Тернопільського національного технічного університету імені Івана Пулюя»

Циклова комісія комп’ютерно

інформаційних технологій

Курсова робота

з дисципліни

«Об’єктно-орієнтоване програмування» на тему:

«Дослідити можливості Python

і бібліотеки pymongo в роботі

з базами даних»

Розробив студент

групи 31-ІПЗ

Павленко Владислав

Керівник проєкту

Баран М. П.

Зборів, 2025 р.

# ****ЗМІСТ****

**Завдання на курсову роботу** .................................................................................. 2

**Перелік умовних позначень** ................................................................................. 4

**Вступ** ....................................................................................................................... 5

1. **Аналіз предметної області, методів та засобів** ........................................... 6
   1. Огляд баз даних та їх класифікація .....................................................6
   2. **MongoDB як представник NoSQL баз даних**  .................................... 6
   3. Бібліотека pymongo для роботи з MongoDB ..................................... 7
   4. Опис предметної області застосунку ................................................. 7
   5. Архітектура та принципи роботи MongoDB ..................................... 8
   6. Порівняння реляційних та нереляційних баз даних ......................... 8
   7. Додаткові можливості pymongo ......................................................... 9
   8. Аналоги MongoDB та альтернативи ................................................ 10
   9. Висновки щодо вибору технологій .................................................. 12
2. **Реалізація програмного забезпечення** ....................................................... 13

2.1. **Постановка задачі** ................................................................................. 13

2.1.1 Завдання до КР ............................................................................ 15

2.1.2 Вхідна інформація ....................................................................... 16

2.1.3 Вихідна інформація ..................................................................... 19  
2.2. **Опис мови програмування** …………………………........................... 22  
2.3. **Опис алгоритму та ідентифікатори** .................................................... 24  
2.4. **Тестування програми** ........................................................................... 27

2.4.1. Методика тестування ................................................................. 28

2.4.2. Модульне тестування ................................................................. 29

2.4.3. Функціональне тестування ........................................................ 30

2.4.4. Результати тестування ............................................................... 31

**Висновки** .............................................................................................................. 34

**Перелік використаної літератури** ....................................................................... 36

**Додатки** ................................................................................................................. 37

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ**

API (Application Programming Interface) − Набір правил і інструментів для взаємодії програмних компонентів.

BSON (Binary JSON) − Бінарний формат даних, що використовується MongoDB для ефективного зберігання документів.

CRUD − Акронім для базових операцій з даними: Create (створення), Read (читання), Update (оновлення), Delete (видалення).

JSON (JavaScript Object Notation) − Текстовий формат обміну даними на основі пар "ключ-значення".

MongoDB − Документоорієнтована NoSQL-база даних, що використовує BSON.

NoSQL − Тип баз даних, які не використовують реляційну модель (на відміну від SQL). Приклади: MongoDB, Redis.

PyMongo − Офіційна бібліотека Python для роботи з MongoDB.

SQL (Structured Query Language) − Мова запитів для роботи з реляційними базами даних (наприклад, MySQL, PostgreSQL).

Unittest − Вбудований модуль Python для автоматизованого тестування коду.

**ВСТУП**

У сучасному світі обробка та збереження великих обсягів даних є основним завданням для багатьох додатків. Традиційні реляційні бази даних, такі як MySQL або PostgreSQL, мають певні обмеження, зокрема складність масштабування та жорстку структуру даних. Саме тому все більшої популярності набувають NoSQL-бази даних, які пропонують гнучкі рішення для зберігання інформації.

MongoDB – одна з найпоширеніших NoSQL-баз даних, яка використовує JSON-подібні документи та забезпечує високу продуктивність і масштабованість. Для роботи з MongoDB у Python використовують бібліотеку pymongo, яка дозволяє легко підключатися до бази даних, виконувати запити та маніпулювати даними.

Дана курсова робота присвячена дослідженню можливостей MongoDB та pymongo на практиці. У межах роботи розроблено консольний застосунок для управління завданнями (тасками), що дозволяє додавати, редагувати, видаляти та переглядати записи. У процесі реалізації буде використано основні CRUD-операції (Create, Read, Update, Delete), які є фундаментальними для роботи з базами даних.

Основна мета роботи – дослідити особливості NoSQL-баз даних, ознайомитися з MongoDB та pymongo, а також реалізувати консольний застосунок для взаємодії з базою даних.

**1** **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ,**

**МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАВДАННЯ**

#### ****1.1 Огляд баз даних та їх класифікація****

У сучасному світі величезна кількість інформації потребує систематизації, швидкого доступу та збереження. Для цього використовуються **бази даних (БД)** – спеціалізовані сховища, в яких інформація організовується за певними принципами.

Бази даних поділяються на два основних типи:

* **Реляційні (SQL)** бази, що зберігають дані у вигляді таблиць (рядки, стовпці). Вони використовують мову **SQL (Structured Query Language)** для запитів[4].
* **Нереляційні (NoSQL)** бази, які не мають фіксованої структури. Вони можуть зберігати інформацію у вигляді **документів, графів, пар "ключ-значення" тощо[10]**.

Реляційні бази (наприклад, MySQL, PostgreSQL) добре підходять для систем з чітко структурованими даними, але вони менш гнучкі у масштабуванні. NoSQL-бази (MongoDB, Firebase, Redis) використовуються для динамічних і гнучких застосунків, де структура даних може змінюватися[10].

#### ****1.2 MongoDB як представник NoSQL баз даних****

MongoDB – одна з найпопулярніших NoSQL баз даних, яка працює за принципом **документного зберігання**. Дані зберігаються у форматі **BSON (Binary JSON)**,що дозволяє легко працювати зі структурованими об'єктами[6].

##### ****Переваги MongoDB:****

* **Гнучка структура** – можна зберігати документи з різною структурою без необхідності змінювати схему.
* **Висока продуктивність** – швидке виконання запитів завдяки індексації.
* **Горизонтальне масштабування** – можливість розподілу даних між серверами.

MongoDB часто використовується у веб-застосунках, фінтех-стартапах, аналітичних системах та мобільних застосунках, де важлива швидкість та масштабованість.

#### ****1.3 Бібліотека pymongo для роботи з MongoDB****

Python має велику кількість бібліотек для взаємодії з базами даних, і для роботи з MongoDB використовується **pymongo** – офіційний драйвер для підключення до MongoDB у Python[5].

##### **Основні можливості pymongo:**

* **Підключення до локальної або віддаленої бази даних (лістинг 1.1)**
* **CRUD-операції** (створення, читання, оновлення, видалення)
* **Робота з індексами та агрегованими запитами**

**Лістинг 1.1 Приклад підключення до Mongo DB**

import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["tasks\_db"] # створюємо або підключаємося до бази

collection = db["tasks"] # створюємо або підключаємося до колекції

Цей код встановлює з’єднання з локальною базою MongoDB та створює базу даних tasks\_db із колекцією tasks.

#### ****1.4 Опис предметної області застосунку****

У межах курсової роботи розроблено **консольний застосунок для управління завданнями (тасками).** Користувач може **додавати, редагувати, видаляти та переглядати завдання**, а дані зберігаються у MongoDB[9].

##### **Основні можливості застосунку:**

* **Додавання нового завдання (назва, опис, статус, термін виконання)**
* **Редагування існуючого завдання**
* **Видалення завдання**
* **Перегляд списку завдань**

Для реалізації було обрано MongoDB, оскільки:

* структура завдань є **динамічною** (можливе додавання нових полів);
* необхідна **швидкість збереження та отримання інформації**;
* немає потреби в складних SQL-запитах – достатньо операцій CRUD.

**1.5 Архітектура та принципи роботи MongoDB**

MongoDB використовує документно-орієнтовану модель зберігання, що робить її дуже гнучкою в порівнянні з реляційними базами[10].

Основні поняття MongoDB:

* Документ – основна одиниця збереження (аналог запису в SQL).
* Колекція – група документів (аналог таблиці в SQL).
* База даних – контейнер для колекцій.
* Shard'ування – розподіл даних між кількома серверами для масштабування.

Документи у MongoDB зберігаються у форматі BSON (Binary JSON), що забезпечує зручну взаємодію з мовами програмування, зокрема Python.

Лістинг 1.2 Приклад структури документа у MongoDB

{

"\_id": ObjectId("650cf123a0b6f2d0e3a12345"),

"title": "Закінчити курсову",

"description": "Написати розділ про MongoDB",

"status": "В процесі",

"due\_date": "2025-03-20"

}

Такий підхід дозволяє додавати або змінювати поля без порушення загальної структури.

**1.6 Порівняння реляційних та нереляційних баз даних**

Щоб краще зрозуміти, чому MongoDB є кращим вибором для даного застосунку, розглянемо основні відмінності:

Таблиця 1.1 – Порівняння реляційних та нереляційних баз даних

|  |  |  |
| --- | --- | --- |
| **Критерій** | **Реляційні БД (SQL)** | **NoSQL БД (MongoDB)** |
| **Структура** | Жорстка (таблиці, рядки, колонки) | Гнучка (JSON-документи) |
| **Масштабування** | Вертикальне (додавання ресурсів серверу) | Горизонтальне (додавання нових серверів) |
| **Запити** | SQL (JOIN, GROUP BY) | JSON-запити (db.collection.find()) |
| **Швидкість** | Висока при складних запитах | Висока при великій кількості записів |
| **Використання** | Фінанси, банківські системи | Веб-додатки, мобільні застосунки, Big Data |

Для керування тасками гнучкість і швидкість MongoDB важливіші за складні зв’язки, тому вибір NoSQL є виправданим[2].

**1.7 Додаткові можливості pymongo**

Окрім базових CRUD-операцій, pymongo підтримує[5]:

* Індекси для прискорення пошуку (create\_index()).
* Агрегаційні запити (aggregate()), що дозволяють виконувати групування та аналіз (лістинг 1.2).
* Транзакції для забезпечення цілісності даних.

Лістинг 1.2 Приклад агрегованого запиту

pipeline = [

{"$group": {"\_id": "$status", "count": {"$sum": 1}}}

]

result = collection.aggregate(pipeline)

for doc in result:

print(doc)

Цей запит дозволяє швидко отримати статистику про завдання у базі[3].

**1.8 Аналоги MongoDB та альтернативи**

Хоча MongoDB – один із найпопулярніших виборів для NoSQL-застосунків, існує багато інших варіантів, які можуть бути кращими залежно від конкретних вимог проекту.

### **Альтернативи MongoDB:**

#### **1. Firebase Firestore**

Firebase Firestore – це хмарне NoSQL-сховище від Google, яке працює за принципом **документно-орієнтованої бази даних**.

Основні особливості:

* **Синхронізація в реальному часі** – оновлення відбувається миттєво на всіх пристроях.
* **Глибока інтеграція з екосистемою Google** – чудово підходить для мобільних додатків (Android, iOS) та веб-застосунків.
* **Автоматичне масштабування** – Google бере на себе керування серверами.
* **Оптимізовано для роботи з Firebase Authentication** – зручно для додатків із авторизацією.

**Варто використовувати:**

* Якщо потрібно працювати з мобільними чи веб-застосунками.
* Якщо потрібна миттєва синхронізація даних між клієнтами.
* Якщо не хочеться налаштовувати сервери.

#### **2. Redis**

Redis – це **супершвидка NoSQL-база даних**, яка працює в оперативній пам’яті (**in-memory database**). Вона використовується переважно для кешування, але також підтримує структури даних, такі як хеш-таблиці, списки, множини тощо.

Основні особливості:

* **Неймовірно швидкий доступ до даних** – завдяки зберіганню в оперативній пам’яті (RAM).
* **Підтримка TTL (Time-To-Live)** – корисно для зберігання тимчасових даних, таких як сесії користувачів або кеш.
* **Підтримка публікації/підписки (Pub/Sub)** – дозволяє реалізовувати повідомлення між сервісами.
* **Підтримка кластеризації** – дозволяє масштабувати базу для високонавантажених проєктів.

**Варто використовувати:**

* Якщо потрібне кешування даних або збереження сесій.
* Якщо необхідна низька затримка в обробці запитів.
* Якщо потрібно реалізувати черги повідомлень чи систему публікацій/підписок.

#### **3. Apache Cassandra**

Cassandra – це **розподілена NoSQL-база даних**, створена для обробки величезних обсягів даних із високою доступністю та відмовостійкістю.

Основні особливості:

* **Децентралізована архітектура** – кожен вузол рівноправний, немає центрального сервера.
* **Висока масштабованість** – легко додаються нові сервери без зупинки роботи системи.
* **Підтримка реплікації між дата-центрами** – ідеально підходить для глобальних проєктів.
* **Підходить для записів у великих обсягах** – використовується компаніями на кшталт Instagram, Netflix і Spotify.

**Варто використовувати:**

* Якщо потрібно обробляти великі обсяги даних із високою доступністю.
* Якщо система повинна працювати навіть при виході з ладу окремих серверів.
* Якщо потрібне горизонтальне масштабування.

Для розробки консольного застосунку MongoDB є найбільш зручним варіантом через простий API та підтримку в Python.

**1.9 Висновки щодо вибору технологій**

На основі аналізу предметної області вибір MongoDB та pymongo є виправданим через:

* Високу гнучкість збереження даних
* Простоту у використанні
* Вбудовані засоби масштабування
* Підтримку CRUD, індексів, агрегацій

Ці можливості роблять MongoDB оптимальним рішенням для консольного менеджера завдань.

**2 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**2.1 Постановка задачі**

Розробка програмного забезпечення для управління списком завдань є важливою задачею у сучасному програмуванні. В умовах високої динаміки сучасного життя, де завдання та робочі обов'язки можуть швидко змінюватися, зберігання та організація завдань стають важливими для підтримки високої продуктивності. Завдяки ефективним інструментам для управління завданнями користувачі можуть зберігати, редагувати та переглядати свої завдання, що дозволяє значно покращити планування часу та контроль за виконанням роботи.

Цей додаток має бути не лише корисним, але й простим у використанні, щоб користувач міг зосередитися на виконанні своїх завдань, а не на складностях взаємодії з програмою. Основною метою є створення ефективного, зручного та інтуїтивно зрозумілого інструменту для управління завданнями, який забезпечить користувачу швидкий доступ до важливої інформації та дозволить надійно зберігати дані. Додаток буде розроблений за допомогою мови програмування Python та використанням бази даних MongoDB.

**Основні вимоги до системи**

Для реалізації програмного забезпечення було визначено низку основних вимог:

* Забезпечення можливості додавання, редагування, перегляду та видалення завдань. Користувач повинен мати змогу легко додавати нові завдання в систему, а також редагувати їх або видаляти в разі потреби. Це включає в себе можливість введення всіх необхідних даних, таких як назва завдання, опис, дата виконання тощо. Всі ці операції повинні бути простими і зрозумілими.
* Реалізація зручного консольного інтерфейсу для взаємодії користувача з додатком. Програма повинна надавати консольний інтерфейс, що дозволить користувачеві швидко виконувати операції, не потребуючи складних навичок програмування або спеціалізованого знання. Інтерфейс повинен бути інтуїтивно зрозумілим і зручним для використання, з чіткими підказками та повідомленнями про помилки.
* Використання MongoDB як основне сховище даних для забезпечення гнучкості та масштабованості. Оскільки завдання можуть бути дуже різними за характером і вимогами до структури даних, важливо використовувати базу даних, яка дозволяє зберігати дані в гнучкому форматі, що легко розширюється. MongoDB є документоорієнтованою базою даних, яка дозволяє працювати з даними у форматі JSON, що є оптимальним для таких задач.
* Реалізація перевірки правильності введених даних. Програма повинна забезпечувати належну перевірку даних, що вводяться користувачем. Це включає перевірку правильності формату введених даних, таких як текстові поля та числові значення. Також слід перевіряти, чи не залишаються обов'язкові поля порожніми, що може призвести до некоректної роботи програми.

**Причини вибору Python та MongoDB**

У виборі технологій для розробки програми було враховано багато факторів, зокрема простота розробки, ефективність роботи з базами даних, а також потреби в масштабованості та зручності роботи з великими обсягами даних.

**Python.** Простота та гнучкість у використанні. Python є однією з найбільш популярних мов програмування завдяки своїй простоті та читабельності коду. Це дозволяє швидко реалізувати функціональність без значних витрат часу на вивчення складних синтаксичних конструкцій.

Велика кількість бібліотек для роботи з базами даних. Python має великий вибір бібліотек для роботи з різними типами баз даних, зокрема з MongoDB. Це дозволяє ефективно працювати з базою даних та інтегрувати її з додатком, використовуючи мінімальні зусилля.

Хороша підтримка спільноти та документація. Оскільки Python є однією з найбільш популярних мов програмування, для неї існує велика кількість ресурсів, включаючи документацію, навчальні посібники та спільноти програмістів, які можуть допомогти при розробці програми.

**Основні переваги MongoDB**

MongoDB використовує документоорієнтовану структуру збереження даних, де дані зберігаються у форматі JSON (JavaScript Object Notation). Це дозволяє зберігати інформацію у зручному і гнучкому форматі, що дає змогу швидко адаптувати базу під зміни в структурі даних без необхідності змінювати всі таблиці або схеми, як у реляційних базах даних.

Завдяки хорошому масштабуванні MongoDB може ефективно працювати з великими обсягами даних. Це є важливою перевагою для розробки програм, де з часом буде додаватися багато записів, для підтримки високої швидкості доступу до даних.

Гнучка схема MongoDB дозволяє додавати нові поля до документів без зміни структури всієї бази даних. Це дає додаткову гнучкість при роботі з різними типами даних, що є важливим при розробці додатків, де структура даних може змінюватися в залежності від вимог.

Загалом, вибір Python та MongoDB забезпечує розробку програмного забезпечення, яке є не лише ефективним та зручним у використанні, але й надійним і масштабованим у процесі подальшого розвитку та вдосконалення.

**2.1.1 Завдання до КР**

Завданням курсової роботи є дослідження можливостей мови програмування Python та бібліотеки pymongo для роботи з базами даних MongoDB. У рамках дослідження буде проведено аналіз основних функцій Python, які дозволяють ефективно взаємодіяти з MongoDB, зокрема для виконання CRUD-операцій (створення, читання, оновлення, видалення даних).

Основними завданнями є:

* Вивчення принципів роботи з базами даних NoSQL, зокрема MongoDB.
* Оцінка переваг і недоліків використання MongoDB порівняно з традиційними реляційними базами даних.
* Розробка програми для управління завданнями з використанням MongoDB як сховища даних.
* Використання бібліотеки pymongo для реалізації з'єднання з MongoDB та виконання основних операцій з даними.
* Оцінка продуктивності та ефективності роботи програми на різних етапах тестування.

Цей проект дозволяє вивчити базові принципи роботи з MongoDB, а також розробити практичні навички з інтеграції Python з базами даних за допомогою pymongo.

#### 2.1.2 Вхідна інформація

Вхідною інформацією для роботи програми є дані, які вводить користувач, а також збережені у базі MongoDB записи. Основні параметри вхідної інформації поділяються на три категорії**:**

1. **введення користувача**
2. **параметри підключення до бази даних**
3. **структура даних**

##### ****1. Введення користувача****

Програма отримує дані через текстовий інтерфейс, у якому користувач вводить команди та відповідні параметри. Основні типи введення:

* **Команди управління тасками**:

1. Get Tasks — виводить список усіх завдань;
2. Get Task by ID — виводить таск по його айді
3. Add Task — додає нове завдання;
4. Mark Task as Complete — позначає завдання як виконане;
5. Update Task — оновлює завдання;
6. Delete Task— видаляє завдання з бази.

* **Помилковий ввід** (програма повинна коректно обробляти некоректні команди).

##### ****2. Параметри підключення до бази даних****

Щоб програма могла працювати з MongoDB, необхідно налаштувати підключення до бази даних. Підключення до MongoDB є важливим етапом, оскільки воно забезпечує взаємодію програми з базою даних, дозволяючи зберігати, змінювати та отримувати завдання. Для цього потрібно вказати кілька важливих параметрів для правильного встановлення з'єднання з сервером MongoDB. Ці параметри можуть включати:

**Адреса сервера MongoDB**. Адреса сервера є основним параметром, який визначає місце розташування MongoDB, до якого буде підключена програма. В залежності від того, де розгорнута ваша база даних, адреса може бути різною:

Локальний сервер MongoDB: Якщо MongoDB працює на вашому комп'ютері або сервері в локальній мережі, то адреса підключення буде виглядати так - mongodb://localhost:27017/.

Тут localhost — це адреса локальної машини, на якій працює сервер MongoDB. 27017 — це порт, на якому MongoDB слухає підключення (за умовчанням MongoDB використовує цей порт)[6].

Хмарний сервер MongoDB (наприклад, MongoDB Atlas): Якщо ви використовуєте хмарну платформу, таку як MongoDB Atlas, для розміщення вашої бази даних, то замість локальної адреси ви отримаєте URL сервера від платформи MongoDB Atlas. Цей URL виглядає як:

Лістинг 2.1 – URL адреса підключення MongoDB

mongodb+srv://<username>:<password>@cluster0.mongodb.net/test?retryWrites=true&w=majority

Тут <username> — ваш логін для підключення до MongoDB Atlas. <password> — ваш пароль. cluster0.mongodb.net — адреса хмарного кластера MongoDB, на якому зберігаються ваші дані. test — ім'я бази даних за умовчанням, яку ви можете змінити на свою.

Підключення до хмарної платформи потребує також налаштування прав доступу та використання SSL для шифрування з'єднання.

**Назва бази даних.** Після вказання адреси сервера необхідно вказати назву бази даних, з якою буде працювати програма. Назва бази даних є унікальним ідентифікатором для вашого набору даних, в якому зберігатимуться всі документи, пов'язані з вашим застосунком. Наприклад – «task\_manager»

У цьому випадку назва бази даних — це task\_manager. Важливо, щоб база даних була створена перед виконанням операцій з нею, або MongoDB автоматично створить її при першій вставці даних.

Примітка: Якщо база даних з таким ім'ям уже існує, MongoDB використовує наявну базу даних. Якщо ж база даних не існує, вона буде створена при першому записі даних.

**Назва колекції**

Колекція в MongoDB є аналогом таблиці в реляційних базах даних. У колекціях зберігаються документи, які являють собою записи з даними. Ваша програма буде працювати з колекцією, де зберігатимуться завдання користувача. Наприклад - «my-tasks»

Колекція my-tasks буде містити всі завдання користувача, кожне з яких буде окремим документом. Важливо, що колекція не обов'язково повинна бути створена вручну — MongoDB створить її автоматично при першому додаванні даних.

##### ****3. Структура вхідних даних****

Кожен запис у базі даних містить наступні атрибути:

* \_id — унікальний ідентифікатор завдання (генерується автоматично MongoDB).
* title — коротка назва завдання.
* description — детальний опис завдання.
* is\_complete — чи виконане завдання ("true", "false").
* folder — папка до якої буде належати завдання.

Лістинг 2.2 Приклад вхідного JSON-запису в MongoDB

{

"\_id": "660b9f1f2f7a4c12a3456789",

"name": "Написати курсову",

"description": "Завершити розділ про тестування",

"is\_complete": "true",

"folder": "Коледж"

}

Завдяки такій структурі MongoDB дозволяє швидко працювати з даними без необхідності визначення жорсткої схеми, що робить її зручною для зберігання завдань.

#### 2.1.3 Вихідна інформація

Вихідною інформацією є результати виконання операцій з даними. Основні типи вихідних даних:

* **Результати виконання команд**:
  + - * Повідомлення про успішне додавання, зміну або видалення завдання.
      * Список завдань при виклику команди Get Tasks.
      * Повідомлення про помилки у разі неправильного введення команд.
* **Формат вихідних даних**:
  + - * Консольний вивід результатів у зручному для користувача вигляді.
      * JSON-структура при взаємодії з базою даних.

##### ****Приклади вихідної інформації****

1. Вивід списку всіх завдань – у цьому прикладі програма виводить повний перелік наявних завдань, кожне з яких містить ідентифікатор, назву, опис, статус виконання та дату створення. Такий формат дозволяє користувачу швидко ознайомитися з поточними справами. (див. рисунок 2.1)

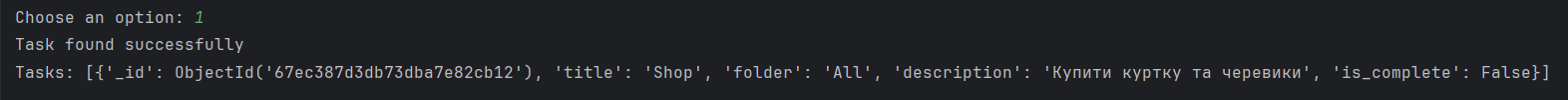


Рисунок 2.1 - Вивід списку завдань

1. **Вивід одного завдання за його ID** – демонструється результат запиту, в якому користувач вводить конкретний ідентифікатор завдання. У відповідь відображається повна інформація лише про обране завдання, що зручно для перегляду або редагування. (див. рисунок 2.2)

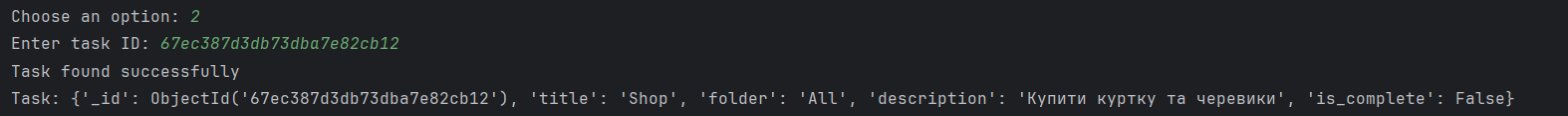


Рисунок 2.2 - Вивід завдання по айді

1. **Вивід при додаванні нового завдання** – показується приклад відповіді системи після створення нового завдання. Програма підтверджує успішне додавання та повертає новостворений об’єкт із усіма введеними даними й присвоєним ID. (див. рисунок 2.3)

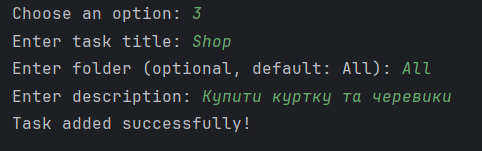


Рисунок 2.3 - Вивід при додаванні нового завдання

1. **Вивід при позначенні завдання як виконаного** – у цьому прикладі завдання змінює свій статус на “виконано”, і програма одразу оновлює це в інтерфейсі. Такий механізм дозволяє відслідковувати прогрес у виконанні справ. (див. рисунок 2.4)

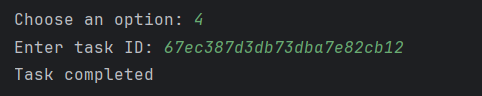


Рисунок 2.4 - Вивід при позначці завдання як виконаного

1. **Вивід при оновленні завдання** – після редагування вже існуючого завдання, наприклад, зміни назви або опису, система повертає оновлений об’єкт. Це дає змогу переконатися, що всі зміни були застосовані коректно. (див. рисунок 2.5)

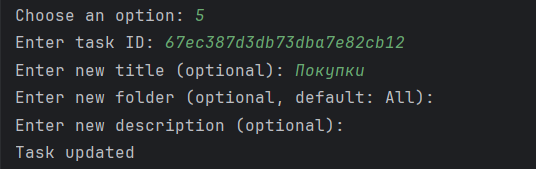


Рисунок 2.5 - Вивід при оновленні завдання

1. **Вивід при видаленні завдання** – коли користувач видаляє завдання, програма підтверджує успішне виконання цієї дії. У деяких випадках повертається коротке повідомлення або статус без тіла відповіді. (див. рисунок 2.6)

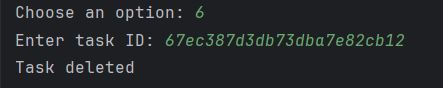


Рисунок 2.6 - Вивід при видаленні завдання

1. **Вивід повідомлення про помилку** – у випадку, коли користувач вводить неіснуючий або некоректний ID, система повідомляє про помилку. Це допомагає уникнути непорозумінь і забезпечує зворотній зв'язок. (див. рисунок 2.7)

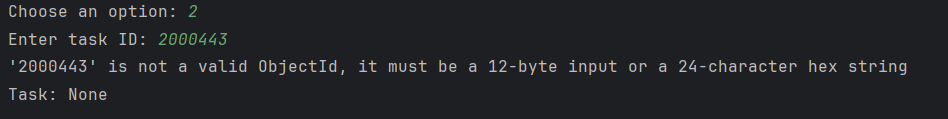


Рисунок 2.7 - Вивід у разі помилки (наприклад, якщо ID невалідний)

Такий формат вихідних даних робить систему зручною у використанні, оскільки забезпечує швидку взаємодію користувача з базою даних, а також інтуїтивно зрозумілий зворотний зв’язок щодо виконаних дій.

### 2.2 Опис мови програмування

Python — це високорівнева мова програмування загального призначення, що має простий і зрозумілий синтаксис. Вона підтримує об'єктно-орієнтоване, процедурне та функціональне програмування, що робить її гнучкою та ефективною для різних завдань. У цьому розділі розглянуто основні особливості Python, його переваги та недоліки, а також причини вибору для реалізації даного проєкту[1, 8].

#### Основні особливості Python:

1. **Простий та зрозумілий синтаксис.** Код, написаний мовою Python, легко читається та розуміється навіть початківцями.
2. **Динамічна типізація.** Python автоматично визначає тип змінної під час виконання програми, що спрощує розробку.
3. **Автоматичне керування пам’яттю**. Вбудований збирач сміття звільняє невикористовувану пам’ять, що покращує ефективність програми.
4. **Багатоплатформність.** Програми, написані на Python, можна запускати на різних операційних системах без змін у коді.
5. **Велика стандартна бібліотека.** Python містить безліч вбудованих модулів для роботи з файлами, мережею, базами даних тощо.
6. **Підтримка багатьох парадигм програмування.** Python дозволяє використовувати різні стилі програмування, що робить його гнучким для різних типів проєктів.

#### Переваги Python:

1. **Швидкість розробки.** Завдяки простому синтаксису та великій кількості бібліотек розробка програм займає менше часу.
2. **Гнучкість.** Python використовується в різних сферах: веб-розробка, аналіз даних, штучний інтелект, автоматизація, наукові обчислення тощо.
3. **Велика спільнота розробників**. Python має активну спільноту, що сприяє швидкому вирішенню проблем і постійному вдосконаленню мови.
4. **Легка інтеграція з іншими мовами**. Python може взаємодіяти з мовами C, C++, Java, що розширює його можливості.

#### Недоліки Python:

1. **Низька швидкодія у порівнянні з компільованими мовами**. Через інтерпретовану природу Python працює повільніше за C++ або Java.
2. **Високе споживання пам’яті**. Автоматичне керування пам’яттю може призводити до її надмірного використання.
3. **Обмеженість у мобільній розробці**. Python менш популярний у розробці мобільних застосунків, оскільки має нижчу продуктивність у порівнянні з Swift або Kotlin.

#### Причини вибору Python для проєкту

Python був обраний для розробки цього проєкту через такі причини:

* **Простота та швидкість розробки**. Python дозволяє швидко створювати прототипи програм, що важливо для навчальних і дослідницьких проєктів.
* **Підтримка роботи з базами даних**. Завдяки бібліотеці pymongo Python забезпечує зручну взаємодію з MongoDB.
* **Широка екосистема бібліотек.** Python має безліч бібліотек для роботи з базами даних, обробки тексту та автоматизації процесів.
* **Кросплатформність.** Розроблену програму можна легко запускати як у середовищі Windows, так і на Linux або macOS.

Таким чином, Python є оптимальним вибором для створення консольного додатка з управління тасками в базі даних MongoDB.

### 2.3 Опис алгоритму та ідентифікатори

Цей розділ містить детальний опис алгоритму роботи консольного додатка, що взаємодіє з базою даних MongoDB для управління тасками. Буде розглянуто загальну логіку програми, основні функції, структуру даних, а також ідентифікатори, які використовуються для керування інформацією.

#### Загальна логіка роботи програми

Програма дозволяє користувачеві виконувати основні операції над списком завдань через консольний інтерфейс. Вона працює за принципом обробки командного введення, де кожна команда відповідає певній CRUD-операції (Create, Read, Update, Delete)[9]. Алгоритм роботи складається з таких основних етапів:

1. Ініціалізація програми та підключення до бази даних MongoDB.
2. Очікування введення команди від користувача.
3. Аналіз введеної команди та її розбір.
4. Виконання відповідної дії:
   * + - Додавання нового завдання.
       - Відображення списку завдань.
       - Оновлення статусу існуючого завдання.
       - Видалення завдання.
5. Виведення користувачеві відповідного повідомлення про результат виконання операції.
6. Очікування нової команди або завершення роботи.

### ****Ініціалізація підключення до бази даних в MongoDB****

Для взаємодії з MongoDB у Python використовується бібліотека pymongo, яка надає зручний інтерфейс для підключення, запису та читання даних.

Процес ініціалізації включає кілька основних етапів:

1. **Підключення до сервера MongoDB** – необхідно встановити з'єднання із сервером, який може працювати локально або в хмарному середовищі, наприклад, через MongoDB Atlas.
2. **Вибір бази даних** – якщо база ще не створена, вона буде створена автоматично при першому використанні.
3. **Створення або вибір колекції** – колекції в MongoDB відповідають таблицям у традиційних реляційних базах даних, але є більш гнучкими через відсутність жорсткої схеми.
4. **Перевірка підключення** – можна отримати список баз або колекцій, щоб упевнитися, що підключення встановлено правильно.
5. **Вставка тестового запису** – MongoDB дозволяє зберігати дані у форматі BSON (схожий на JSON), тому тестова вставка допоможе перевірити роботу бази.
6. **Закриття підключення** – хоча MongoDB самостійно управляє з’єднаннями, рекомендується закривати його після завершення роботи для оптимального використання ресурсів.

Детальний код для реалізації цього процесу наведено в **Додатку A**, де описані всі необхідні кроки для ініціалізації та перевірки підключення до бази даних.

#### Алгоритм обробки команд

Програма аналізує введену команду та виконує відповідну дію. Основний алгоритм виглядає так:

1. Запуск програми – ініціалізується підключення до MongoDB.
2. Введення команди користувачем – користувач вводить команду у форматі цифри з меню (див. Додаток Б).
3. Обробка введення – програма розпізнає команду та розбиває її на аргументи (див. Додаток В).
4. Виконання дії – програма робить відповідний запит до бази даних(див. Додаток Г)[7].
5. Виведення результату – повідомлення про успішне виконання команди або помилку (див. Додаток Ґ).
6. Опис ідентифікаторів
7. Для ефективного керування даними в базі використовується кілька ідентифікаторів:

\_id — унікальний ідентифікатор завдання (генерується автоматично MongoDB).

title — коротка назва завдання.

description — детальний опис завдання.

is\_complete — чи виконане завдання ("true", "false").

folder — папка до якої буде належати завдання.

1. Структура даних у базі MongoDB знаходиться у Лістингу 2.2

### ****Блок-схема роботи програми****

Для кращого розуміння логіки роботи консольного додатка розроблено **блок-схему**, яка графічно відображає алгоритм його функціонування. Блок-схема показує основні етапи виконання команд, а також умови переходу між ними.

На початку роботи програма ініціалізує підключення до бази даних MongoDB. Далі очікується введення команди користувачем, яка може відповідати одній з CRUD-операцій: створення нового завдання, перегляд списку завдань, оновлення існуючих даних або видалення записів.

Кожен введений запит проходить перевірку:

* Якщо команда є коректною, вона передається для обробки.
* У разі некоректного введення програма повідомляє про помилку і запитує команду повторно.

Якщо команда стосується змін у базі даних, відповідний запит виконується через MongoDB. Після цього користувач отримує повідомлення про успішне виконання дії або помилку.

Окремий логічний блок відповідає за завершення роботи програми. Якщо користувач вводить команду виходу, програма закриває з'єднання з базою даних, виводить повідомлення про завершення роботи і припиняє виконання.

Блок-схема роботи програми детально представлена у **Додатку Д**, де наочно відображені всі етапи обробки команд, перевірки введених даних та взаємодії з базою даних.

Цей розділ детально описує, як працює програма, її алгоритм і основні ідентифікатори, що забезпечують зручне управління завданнями у базі даних.

**2.4 Тестування програми**

Тестування програмного забезпечення є одним із найважливіших етапів розробки. Воно дозволяє не тільки перевірити коректність виконання функцій програми, але й виявити помилки, які можуть виникати під час її роботи. Крім того, тестування допомагає оцінити ефективність і продуктивність програмного продукту, що є важливим для забезпечення його стабільної роботи на всіх етапах експлуатації. Адже навіть найкраще спроектоване програмне забезпечення може містити непомічені помилки, що може призвести до втрати даних, некоректної роботи або навіть збою системи.

Цей розділ містить детальний опис процесу тестування програми, включаючи використані методики, інструменти, конкретні тестові сценарії та результати, які дозволяють переконатися в коректності та надійності програми.

Тестування програмного забезпечення включає в себе різні підходи та методи, кожен з яких спрямований на перевірку конкретних аспектів функціональності. Деякі з найбільш важливих аспектів, які потрібно перевірити під час тестування, включають правильність виконання функцій, взаємодію з користувачем, коректність обробки помилок та продуктивність програми.

**2.4.1 Методика тестування**

Для ефективного тестування програми були застосовані два основних підходи: модульне тестування та функціональне тестування. Кожен з цих підходів дозволяє детально перевірити різні частини програми і забезпечити її стабільну роботу в реальних умовах[3]. Весь код моєї роботи збережено на моєму гітхаб[11].

**Модульне тестування**

Модульне тестування є важливим етапом перевірки окремих функцій або модулів програми. Основна мета цього тестування — переконатися, що кожен компонент програми працює належним чином. Наприклад, при перевірці функцій, які виконують CRUD-операції (створення, читання, оновлення, видалення даних), тестувальник має перевірити, чи кожна операція працює коректно при різних умовах вводу. Це дозволяє виявити можливі дефекти на ранніх етапах і зменшити ризик виникнення проблем у майбутньому.

Для модульного тестування використовувалася бібліотека unittest, що є вбудованим модулем Python для автоматичного тестування. Вона дозволяє легко писати і запускати тести, а також порівнювати фактичні результати з очікуваними. Таким чином, ми можемо швидко перевірити, чи кожен окремий компонент працює згідно з вимогами[3].

**Функціональне тестування**

Функціональне тестування спрямоване на перевірку того, як програма взаємодіє з користувачем. В цьому випадку тестувальник перевіряє, чи правильно працюють всі функціональні можливості програми, такі як введення даних користувачем, обробка цих даних програмою та відображення результатів на екрані. Всі ці етапи повинні бути протестовані з урахуванням різних сценаріїв користувацьких дій[3].

**Для тестування програми були використані наступні інструменти**

unittest — вбудований модуль для тестування в Python, який дозволяє автоматизувати перевірку функцій і методів програми. Модуль автоматично порівнює фактичні результати з очікуваними, що дозволяє швидко виявити помилки.

#### Тестові сценарії — створені на основі реальних умов використання програми. Ці сценарії перевіряють різні варіанти вводу, обробки даних і виведення результатів.

#### 2.4.2 Модульне тестування

Модульне тестування є основою для перевірки правильності роботи окремих функцій програми. У цьому випадку основна увага зосереджена на функціях, які взаємодіють з базою даних, оскільки правильність роботи з даними є критично важливою для забезпечення стабільної роботи програми.

Тестування було проведено за допомогою бібліотеки unittest, яка дозволяє перевіряти функції без необхідності вручну запускати програму. Усі тести були написані таким чином, щоб перевірити базові CRUD-операції на різних даних.

Приклад тесту для модульного тестування знаходиться в Лістинг А.6 (див. Додаток Е).

#### 2.4.3 Функціональне тестування

Функціональне тестування було проведене шляхом взаємодії з консольним інтерфейсом програми. Тестування охоплює такі сценарії:

Таблиця 2.1 Результати функціонального тестування

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Опис тесту** | **Вхідні дані** | **Очікуваний результат** | **Фактичний результат** |
| 1 | Отримання списку завдань | - | Відображається список завдань | Відповідає очікуваному |
| 2 | Отримання завдання за ID | ID існуючого завдання | Відображаються деталі завдання | Відповідає очікуваному |
| 3 | Додавання нового завдання | Назва, папка, опис | Завдання успішно додано | Відповідає очікуваному |
| 4 | Оновлення завдання | ID, нові дані | Завдання оновлено | Відповідає очікуваному |
| 5 | Позначення завдання як виконане | ID існуючого завдання | Статус змінюється на "True" | Відповідає очікуваному |
| 6 | Видалення завдання | ID існуючого завдання | Завдання видалено з бази | Відповідає очікуваному |
| 7 | Отримання неіснуючого завдання | Невірний ID | Вивід повідомлення про помилку | Відповідає очікуваному |
| 8 | Додавання завдання з некоректними даними | Порожні поля | Вивід помилки валідації | Відповідає очікуваному |

#### 2.4.4 Результати тестування

Після виконання всіх тестових сценаріїв і перевірки кожної функції програма продемонструвала загалом задовільні результати. Всі основні функціональні можливості програми працюють коректно, а взаємодія з користувачем та базою даних не призводить до збоїв або помилок. Однак під час тестування були виявлені деякі незначні проблеми, які потребували додаткового виправлення для забезпечення більш стабільної роботи програми в реальних умовах.

Під час тестування було виявлено кілька помилок, що стосуються обробки введених даних і взаємодії програми з користувачем. Деякі з цих помилок виникли через некоректне введення користувачем або відсутність перевірки на наявність необхідних даних у базі.

**1. Помилка при введенні некоректного ID**

Одна з найбільших проблем, яка була виявлена під час тестування, полягає в тому, що програма не перевіряла коректність введеного ID перед виконанням операцій оновлення та видалення. Користувач міг ввести ID, який не існує в базі даних, що призводило до спроби оновлення або видалення неіснуючого запису.

Рішення: Для виправлення цієї проблеми було додано механізм перевірки існування запису в базі даних перед виконанням операцій. Якщо введений ID не відповідає жодному з наявних записів, програма виводить відповідне повідомлення про помилку, не намагаючись виконати операцію.

**2. Некоректна валідація введених даних при додаванні завдання**

Ще однією проблемою було те, що програма не завжди коректно перевіряла введення користувача при додаванні нових завдань. Наприклад, якщо користувач залишав поле для опису завдання порожнім, програма не виводила попередження і дозволяла додати завдання з неповними даними.

Рішення: Було додано додаткову перевірку валідації для кожного поля введення. Тепер програма виводить повідомлення про помилку, якщо користувач залишає будь-яке обов'язкове поле порожнім. Це дозволяє уникнути введення неповних даних і забезпечує більш коректну роботу програми.

**3. Проблеми з обробкою помилок при взаємодії з базою даних**

Іншою проблемою, яка була виявлена під час тестування, є обробка помилок, що виникають під час взаємодії з базою даних. Якщо під час виконання операцій з базою даних виникала помилка, програма не завжди правильно обробляла її і виводила загальне повідомлення про помилку, що могло заплутати користувача.

Рішення: Для виправлення цієї проблеми була додана обробка помилок на рівні кожної функції, що взаємодіє з базою даних. Тепер у разі помилки програма виводить чітке повідомлення, яке вказує на тип помилки, що виникла, наприклад, "Помилка при з'єднанні з базою даних" або "Невірний запит до бази".

**4. Проблеми з тестуванням на різних операційних системах**

При тестуванні програми на різних операційних системах було виявлено, що на деяких системах програма не може коректно обробляти деякі специфічні символи в даних, такі як символи національних алфавітів або спецсимволи

Рішення: Було додано додаткові перевірки для коректної обробки таких символів. Тепер програма правильно працює з текстовими даними, що містять не лише стандартні латинські літери, а й символи інших мов.

Враховуючи, що програма проходила всі основні тести без помилок, можна зробити висновок, що вона працює стабільно. Однак, варто зазначити, що деякі проблеми, такі як неправильна валідація введених даних, можуть призвести до негативного користувацького досвіду, якщо вони не будуть виправлені. Тому постійне покращення механізмів обробки помилок та валідації введення є важливою частиною подальшого розвитку програми.

Продуктивність програми також була перевірена під час тестування великих обсягів даних. Зокрема, було проведено тестування з великою кількістю завдань, що повинно було перевірити, як програма працює з великими наборами даних. В результаті тестування було встановлено, що програма здатна ефективно працювати навіть з великими обсягами інформації, не спостерігається значного сповільнення часу виконання операцій.

Після всебічного тестування можна сказати, що програма працює стабільно і коректно виконує всі основні функції. Однак виявлені незначні помилки, такі як некоректна валідація даних і обробка помилок, були виправлені для підвищення зручності використання і стабільності роботи програми.

**ВИСНОВКИ**

У ході виконання курсової роботи було досліджено можливості мови програмування Python та бібліотеки pymongo для роботи з базами даних. Було розроблено консольний додаток для управління завданнями, що дозволяє виконувати основні операції з базою даних MongoDB: створення, читання, оновлення та видалення записів (CRUD-операції).

Під час виконання роботи було реалізовано такі основні завдання:

1. **Досліджено теоретичні основи роботи з базами даних**.  
   Було вивчено особливості NoSQL-баз даних та їх переваги у порівнянні з реляційними базами. MongoDB було обрано як основну СУБД через її гнучкість, масштабованість та зручність роботи з JSON-подібними документами.
2. **Обґрунтовано вибір мови програмування**. Python було обрано завдяки його простоті, широкому набору бібліотек для роботи з базами даних та можливості швидкої розробки. Бібліотека pymongo забезпечила зручну взаємодію з MongoDB.
3. **Розроблено консольний додаток**. Було створено програму, що дозволяє користувачам додавати нові завдання, переглядати список завдань, оновлювати їх статус та видаляти непотрібні записи.
4. **Реалізовано валідацію введених даних**. Використання бібліотеки pydantic дозволило перевіряти правильність введених даних перед їх записом у базу, що допомогло уникнути помилок при роботі з програмою.
5. **Виконано тестування програмного забезпечення.** Для перевірки коректності роботи програми було проведено тестування основних сценаріїв взаємодії з базою даних. Тестування показало, що програма стабільно виконує всі передбачені операції, а також коректно обробляє помилковий ввід користувача.

Розроблений додаток є базовою версією системи керування завданнями, яка може бути значно розширена. Подальший розвиток може включати:

* **Розробку графічного інтерфейсу**, що зробить програму більш зручною для користувачів.
* **Додавання можливості роботи в багатокористувацькому режимі,** з підтримкою авторизації та персоналізованого списку завдань.
* **Інтеграцію з іншими сервісами**, такими як Google Calendar або Telegram-бот для сповіщень про завдання.
* **Розширення функціоналу**, наприклад, додавання дедлайнів, категорій завдань та можливості пріоритизації.

Виконана курсова робота показала, що Python є потужним інструментом для роботи з базами даних, а використання MongoDB дозволяє зручно зберігати та обробляти інформацію у вигляді документів. Розроблений консольний додаток демонструє основні принципи роботи з NoSQL-базами даних та може бути використаний як основа для подальших проєктів у цій сфері.

## ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. **Грінченко Д. М.** Програмування мовою Python. Практичний курс. – Київ: Наукова думка, 2020. – 256 с.
2. **Васильєв А.** Python і робота з базами даних. – Київ: Техніка, 2020. – 198 с.
3. **McKinney W.** Python for Data Analysis. – O’Reilly Media, 2022. – 550 p.
4. **Beaulieu A.** Learning SQL. – O’Reilly Media, 2020. – 377 p.
5. PyMongo 4.11.3 Documentation [Електронний ресурс]. – Режим доступу: <https://pymongo.readthedocs.io/en/stable/> - Дата звернення: 16.03.2025
6. **MongoDB, Inc.** Офіційна документація MongoDB [Електронний ресурс]. – Режим доступу: <https://www.mongodb.com/docs/> - Дата звернення: 16.03.2025
7. **Stack Overflow.** Community discussions on Python and MongoDB [Електронний ресурс]. – Режим доступу: <https://stackoverflow.com/> - Дата звернення: 16.03.2025
8. **W3Schools.** Python Tutorial [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/python/ - Дата звернення: 14.03.2025
9. **GeeksForGeeks.** Python and MongoDB [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/python-mongodb/
10. **Martinez J.** MongoDB Basics. – Apress, 2021. – 250 p.
11. **GitHub репозиторій.** Код програми [Електронний ресурс]. – Режим доступу: https://github.com/Vladislav-Pavlenko/collage/tree/main/python/todo - Дата звернення: 11.03.2025

ДОДАТКИ

**Додаток А**

Лістинг - Ініціалізація підключення бази даних MongoDB

from pymongo.server\_api import ServerApi

from pymongo import MongoClient

import os

from dotenv import load\_dotenv

load\_dotenv()

db\_user = os.getenv('MONGODB\_USER')

db\_password = os.getenv('MONGODB\_PASSWORD')

db\_url = os.getenv('MONGODB\_URL')

db\_name = os.getenv('MONGODB\_DB')

uri = f"mongodb+srv://{db\_user}:{db\_password}@{db\_url}/ {db\_name}?retryWrites=true&w=majority&appName=My-contacts"

try:

client = MongoClient(uri, server\_api=ServerApi('1'))

db = client['my-tasks']

#### tasks\_collection = db['tasks']

#### print("Successfully connect to MongoDB")

#### except Exception as err:

#### print(err)

**Додаток Б**

Лістинг - Введення команди користувачем

while True:

display\_menu()

choice = input("Choose an option: ")

if choice == "1":

get\_tasks()

elif choice == "2":

get\_task\_by\_id()

elif choice == "3":

add\_task()

elif choice == "4":

mark\_as\_complete()

elif choice == "5":

update\_task()

elif choice == "6":

delete\_task()

elif choice == "7":

print("Exiting...")

break

else:

print("Invalid choice, please try again.")

**Додаток В**

Лістинг - Обробка введеної команди користувачем

def get\_tasks():  
 tasks = task.getTasks()  
 print("Tasks:", tasks)  
  
def get\_task\_by\_id():  
 task\_id = input("Enter task ID: ")  
 if task\_id:  
 try:  
 task\_data = task.getTaskById(task\_id)  
 print("Task:", task\_data)  
 except Exception as e:  
 print("Error:", str(e))  
  
def add\_task():  
 title = input("Enter task title: ")  
 folder = input("Enter folder (optional, default: All): ") or "All"  
 description = input("Enter description: ")  
   
 data = {  
 "title": title.strip(),  
 "folder": folder.strip(),  
 "description": description.strip(),  
 }  
   
 try:  
 valid\_task = TaskModel(\*\*data)  
 task.addTask(valid\_task.dict())  
 print("Task added successfully!")  
 except ValidationError as e:  
 print("Validation Error:", e.json())  
  
def mark\_as\_complete():  
 task\_id = input("Enter task ID: ")  
 if task\_id:  
 result = task.markTaskIsComplete(task\_id)  
  
def update\_task():  
 task\_id = input("Enter task ID: ")  
 title = input("Enter new title (optional): ").strip()  
 folder = input("Enter new folder (optional, default: All): ").strip() or "All"  
 description = input("Enter new description (optional): ").strip()  
  
 data = {k: v for k, v in {"title": title, "folder": folder, "description": description}.items() if v}  
  
 try:  
 valid\_task = UpdateTaskModel(\*\*data)  
 task.updateTaskById(valid\_task.dict(), task\_id)  
 except ValidationError as e:  
 print("Validation Error:", e.json())  
  
def delete\_task():  
 task\_id = input("Enter task ID: ")  
 if task\_id:  
 task.deleteTaskById(task\_id)

**Додаток Г**

Лістинг - Виконання запиту за даними

def getTasks():

return list(tasks\_collection.find())

def getTaskById(id):

task = tasks\_collection.find\_one({"\_id":ObjectId(id)})

if task == None:

raise Exception("Not found Task")

return task

def addTask(data):

tasks\_collection.insert\_one(data)

def markTaskIsComplete(id):

task = tasks\_collection.find\_one\_and\_update({"\_id": ObjectId(id)}, {"$set": {"is\_complete": True}})

if task == None:

raise Exception("Not found Task")

def updateTaskById(task, id):

def validate\_and\_clean\_data(data):

return {key: value for key, value in data.items() if value}

valid\_task = validate\_and\_clean\_data(task)

task = tasks\_collection.find\_one\_and\_update({"\_id": ObjectId(id)}, {"$set": valid\_task})

if task == None:

raise Exception("Not found Task")

def deleteTaskById(id):

task = tasks\_collection.find\_one\_and\_delete({"\_id": ObjectId(id)})

if task == None:

raise Exception("Not found Task")

return task

**Додаток Ґ**

Лістинг - Виведення результату виконання команди

def getTasks(self):  
 task = getTasks()  
 print("Task found successfully")  
 return task  
def getTaskById(self, id):  
 try:  
 task = getTaskById(id)  
 print("Task found successfully")  
 return task  
 except Exception as error:  
 print(error)  
def addTask(self, data):  
 addTask(data)  
  
def markTaskIsComplete(self, id):  
 try:  
 task = markTaskIsComplete(id)  
 print("Task completed")  
 return task  
 except Exception as error:  
 print(error)  
def updateTaskById(self, task, id):  
 try:  
 updateTaskById(task, id)  
 print("Task updated")  
 except Exception as error:  
 print(error)  
def deleteTaskById(self, id):  
 try:  
 deleteTaskById(id)  
 print("Task deleted")  
 except Exception as error:  
 print(error)

**Додаток Е**

Лістинг - Тестування роботи з завданнями

import unittest

from pydantic import ValidationError

from python.todo.models.Task import Task

from python.todo.utils.validateData import TaskModel, UpdateTaskModel

from bson import ObjectId

class TestTaskManager(unittest.TestCase):

def setUp(self):

self.task = Task()

self.sample\_task = {

"title": "Test Task",

"folder": "Work",

"description": "This is a test task"

}

valid\_task = TaskModel(\*\*self.sample\_task)

self.task\_id = self.task.addTask(valid\_task.dict())

def test\_get\_tasks(self):

tasks = self.task.getTasks()

self.assertTrue(isinstance(tasks, list))

def test\_get\_task\_by\_id(self):

task\_data = self.task.getTaskById(self.task\_id)

self.assertIsNotNone(task\_data)

self.assertEqual(task\_data["title"], "Test Task")

def test\_add\_task(self):

new\_task = {

"title": "New Task",

"folder": "Personal",

"description": "Another test task"

}

valid\_task = TaskModel(\*\*new\_task)

task\_id = self.task.addTask(valid\_task.dict())

self.assertIsInstance(task\_id, str)

self.assertTrue(ObjectId.is\_valid(task\_id))

def test\_update\_task(self):

update\_data = {"title": "Updated Task", "folder": "Updated Folder", "description": "Updated Description"}

valid\_update = UpdateTaskModel(\*\*update\_data)

self.task.updateTaskById(valid\_update.dict(), self.task\_id)

updated\_task = self.task.getTaskById(self.task\_id)

self.assertEqual(updated\_task["title"], "Updated Task")

def test\_mark\_task\_as\_complete(self):

self.task.markTaskIsComplete(self.task\_id)

updated\_task = self.task.getTaskById(self.task\_id)

self.assertEqual(updated\_task.get("status"), "Complete")

def test\_delete\_task(self):

self.task.deleteTaskById(self.task\_id)

deleted\_task = self.task.getTaskById(self.task\_id)

self.assertIsNone(deleted\_task)

def tearDown(self):

self.task.deleteTaskById(self.task\_id)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**Додаток Д**

Рисунок - Блок-схема роботи програми

